IN-19
185506
49P

NASA
Reference
Publication
1315

1993

NASA Airborne Satellite Instrumentation Calibrator (NASIC) Technical Reference

John L. Ward and Gerry McIntire

NASA Reference Publication 1315

1993

NASA Airborne Satellite Instrumentation Calibrator (NASIC) Technical Reference

John L. Ward
Wallops Flight Facility
Wallops Island, Virginia

Gerry McIntire
Computer Sciences Corporation
Wallops Island, Virginia

National Aeronautics and Space Administration

Scientific and Technical Information Branch

TABLE OF CONTENTS

[illegible]

1. INTRODUCTION

This publication describes the design, operation and function of an Ebert-Fastie monochronomator, which by means of a moveable diffraction grating, becomes a visible and near-infrared spectrometer used to calibrate satellite-borne instruments by high altitude underflights in the NASA ER-2.

The goal is to provide the reader with enough information to operate, repair and modify the spectrometer and associated subsystems hereinafter referred to as NASIC, an acronym for **NASA Airborne Satellite Instrument Calibration** (Project). Included are all hardware schematics, software source listings, and a cursory analysis of test data taken from viewing a calibration sphere before turning the instrument over to Code 925 at the NASA Goddard Space Flight Center in March of 1993.

2. HISTORY OF THE NASIC INSTRUMENT

The precursor to the NASIC was called the Ocean Color Scanner (OCS). The OCS Instrument was built in the early 1980's by NOAA and transferred to NASA in 1988. The instrument was flown aboard the NASA/Lewis Lear jet and subsequently on the NASA ER-2 based at AMES Research Center at Moffett Field, California. The name of the instrument was changed to NASIC shortly after it was transferred to NASA.

The OCS data system consisted of an HP 9825B computer. Program code and data were stored on a magnetic tape cartridge. All I/O and A/D conversion was accomplished by modules which plugged into the back of the 9825B. All code was written in BASIC.

In 1991, a decision was made to modernize all of the NASIC electronics and data system but to leave the grating, optics, and drive mechanisms intact. It was hoped that replacing the HP computer with a PC-compatible computer would increase the system reliability, performance, and adaptability.

System reliability and adaptability have certainly improved, but we have discovered that the system performance is driven by mechanical considerations (jitter in the chain driven grating) that can not be fixed with new electronics and software. This issue will be discussed further in the data analysis section.

3. THEORY OF OPERATION (HARDWARE)

The NASIC instrument measures light between 400 and 1035 nanometers. The spectral resolution is determined by the width of the entrance and intermediate slit together with the lines/mm of the grating. The instrument will fly with slits yielding either a 7 nm or 14 nm resolution.

NASIC is comprised of three major subsystems, the spectrometer, the turret assembly, and the data system. The spectrometer is a cylindrical tube about 16 inches long which contains a telescope and associated optics, a monochromator, a chain driven diffraction grating, and the A/D circuitry. The spectrometer is mounted in the turret assembly which is nothing more than a motorized cradle which allows the telescope and the entrance slit of the spectrometer to be pointed at any desired azimuth and elevation. Finally, the data system is a rack-mounted STD bus containing a PC compatible computer. The computer controls the movement of the turret assembly, the acquisition of spectral and other ancillary data, and the recording of all the data onto a solid state hard drive.

All schematics are included in the appendix.

3.1 Design Considerations

The NASIC is designed to fly onboard a NASA ER-2 at 70,000 feet. Any instrument that flies on the ER-2 must be able to operate without human intervention and at pressure altitudes exceeding 30,000 feet. Given that the NASIC is a passive instrument, designing it to work without human intervention is a relatively simple matter. One simply creates a command file which is called by the "autoexec.bat" file at boot time.

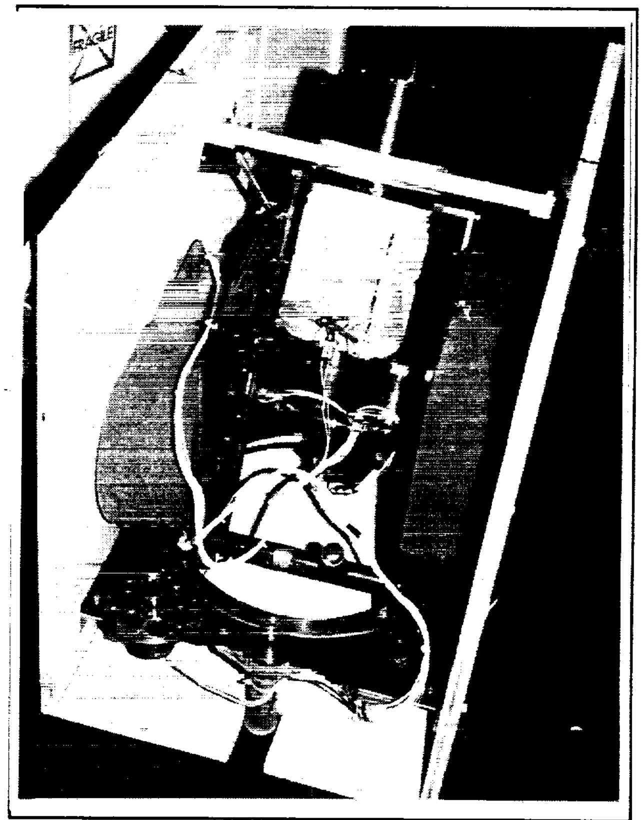


Figure 1. NASIC Spectrometer mounted in the turret.

The considerations for operating at high altitudes present greater challenges. The NASIC data rate is very low. Each spectral scan takes approximately 5 seconds and the data system records a maximum of only 2000 bytes of information per scan. Given the low data rate, recording on a hard drive is the logical choice. The problem is that the heads of a hard drive are designed to have aerodynamic lift. Hard drive heads typically do not have sufficient lift at pressure altitudes greater than 15,000 feet. In the past, pressure boxes were built to house pressure sensitive devices such as hard drives and tape drives. Fortunately, recent advances in EEPROM technology have allowed us to opt for a solid state IDE hard drive, thereby eliminating the need for a pressure box of any kind.

There are a host of other potential problems when operating at high altitudes that we have had the good fortune of not having to learn about, hence they will not be discussed. Engineers from Lockheed and NASA Ames review all instrument designs before being allowed to fly on the ER-2.

3.2 Spectrometer

3.1.1 Ebert-Fastie Double Monochronomator

Shown below is a simplified diagram of the monochronomator. Light enters the entrance slit at the top left and bounces off the concave mirror and onto the grating. The light is then dispersed and reflected by the grating. The light then bounces off the mirror once again and is focused onto the exit slit. As the grating is rotated, the wavelength of light that is focused at the exit slit varies.

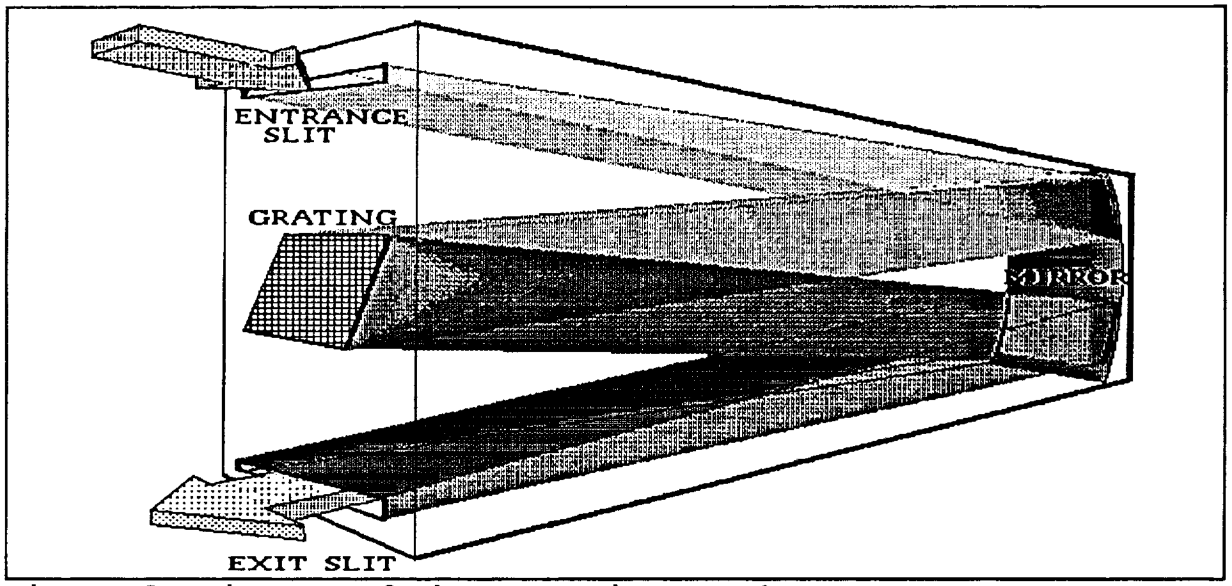


Figure 2. Diagram of Ebert-Fastie Monochronomater.

The diffraction grating is driven by a chain gear which in turn is driven by a 28 Volt stepper motor. It has been found that the chain drive introduces a mechanical jitter which translates into an uncertainty or noise in the determination of the wavelength of the light being measured. This mechanical jitter is in fact the dominant source of error in the instrument and this jitter in effect determines the instrument signal to noise.

3.1.2 Photo-Diode and Analog to Digital Conversion Circuitry

Monochromatic light falls upon a silicon photodiode (HUV4000B). This detector has a built-in FET amplifier, and a gain-bandwidth product of 100 KHz. The output of the photodiode is filtered and split into two channels, a high gain channel and a low gain channel. The high gain channel has twice the gain of the low gain channel. These two channels then feed two of eight inputs of an analog multiplexer (MUX). Other MUX inputs are a reference zener diode, two thermistors, supply voltages, and ground. Each of these inputs is digitized with a 16-bit converter with serial output. The serial data stream exits the rear of the spectrometer as buffered TTL and is converted to RS232 on a board on the turret assembly before being sent to the data system.

A Peltier device is thermally bonded to the back of the photodiode to maintain a constant temperature of roughly 15 degrees Celsius. The Peltier device is controlled by a third party controller which is mounted in the rack along with the data system.

3.3 Turret Assembly

The spectrometer is mounted in the turret assembly which allows the telescope to be pointed in the desired direction. One DC motor drives a ring gear which controls the azimuthal position of the telescope while another DC motor drives a threaded rod which controls the elevation of the telescope. Both DC motors are driven by the computer and use optical shaft angle encoders as feedback to determine the proper positions. The turret assembly also houses a board to buffer data and address lines going to and from the data system and spectrometer.

3.4 Data System

The data system is a rack mount STD bus which contains the following:

-> AT Compatible Motherboard (2 cards)
-> VGA Display Controller Card
-> Solid State (EEPROM based) IDE Hard Drive
-> Stepper Motor Controller
-> Custom Interface Card

The custom interface card has circuitry for the following:

-> Control the spectrometer MUX and A/D.
-> Read the A/D serial data stream.
-> Control the turret azimuth and elevation motors.
-> Read the azimuth and elevation encoders.

3.5 Hardware Changes/Enhancements

Listed below is a summary of hardware changes and enhancements made to the instrument.

Analog to Digital Circuitry - The old system used a 12-bit A/D. The A/D was in a module on the back of the HP computer so that analog signals had to be sent over some length of cable before being digitized. The pre-amp circuitry in the spectrometer was hand-wired on perforation board. The new A/D is 16 bit and is on a 4-layer printed circuit board inside the spectrometer. By digitizing the output of a reference zener diode we find that the A/D has an effective resolution of 14.5 bits at DC. The spectrometer now sends a digital data stream to the data system for recording instead of sending analog signals to the data system for conversion and storing.

Photodiode Channels - The old system had two photodiode channels, a low gain channel and a high gain channel, but only one channel could be recorded on any given flight. The new system also has two channels but both channels are always recorded.

Azimuth/Elevation Circuitry - The old system had potentiometers coupled to the azimuth gear and elevation screw. Azimuth and elevation were determined by digitizing the voltage drop across the potentiometers. This seemed unnecessarily complicated. The potentiometers were replaced with optical shaft angle encoders. Basic trigonometric rules were used to derive a relation between encoder counts and azimuth/elevation angles. The Principal Investigator has suggested using a straight line approximation relate encoder counts to azimuth/elevation angles.

Computer - The old system used an HP 9823B computer. The new system uses a rack mount STD bus system with a PC compatible computer card.

Mass Storage - The old system used tape for both program memory and for storing the instrument data. The new system uses solid state non-volatile memory for both program memory and for storing instrument data.

Peltier Control - The old system used a hand-wired board to control the Peltier device. There was no way to easily change any of the controller parameters. The new system uses a commercial controller with the ability to set the temperature and feedback loop parameters arbitrarily.

Relays - The old system used classic coil and contact relays. The new system uses potted solid state relays.

4.0 THEORY OF OPERATION (SOFTWARE)

All flight software was written in "C" using Borland compilers. All software is contained in one source file which includes the main routine, all subroutines, and all header information. The name of the source file was called "run.c" when it was shipped to Greenbelt. Given that it was the intention of the Principal Investigator to modify the software to suit his needs, one should not assume that the current software is exactly as described in this document. In addition to the flight code, "run.c", a general

purpose diagnostic program was written, the executable of which is called "nasic.exe". This program is used principally for testing the spectrometer. Any MUX channel can be selected and the data viewed in graphical form. In addition, the operation of the turret motors and encoders can be checked with this program.
The program "dcode.c" is included to demonstrate how to decode the NASIC data recorded on ER-2 satellite underflights.

4.1 Command File Structure

The NASIC data system must operate without human intervention. Shortly after takeoff the ER-2 pilot applies power to NASIC. At that time the NASIC data system boots and the flight executable is called by the "autoexec.bat" file. The flight code reads a file called "nasic.cmd" which tells the instrument how to operate. The file "nasic.cmd" is an ascii text file which contains up to six command entries. This means that on any given flight the instrument can fly up to six flight lines. On each flight line the command file tells the instrument the azimuth and elevation in degrees to orient the telescope, the time to begin taking data, and the number of scans to record. The last two entries in the command file set the number of bins per scan and the number of pitch and roll samples per scan for all the flight lines. At this time the instrument is not configured to read the aircraft pitch and roll gyros, so the number of pitch and roll samples per scan entry is ignored.

A command file can be created using any text editor and must be called "nasic.cmd". Shown below is an example command file. Labels could be added to each entry of the command file to make it easier to read or modify if someone takes the time to modify the subroutine "ReadCmdFile()" to strip out the label information during run time.
Example nas.cmd File

```
10:00:00
55
15
38
10:30:00
20
10
38
10:48:50
13
13
38
12:45:05
120
20
18
13:05:00
```

```
90
30
18
13:30:00
110
10
18
200
20
```

The first line (10:00:00) tells the instrument in hours, minutes, and seconds when to begin the first flight line. The second line specifies the telescope azimuth (55 degrees) and the third specifies the instrument elevation(15 degrees). The fourth line specifies the number of spectral scans to record (38). This sequence repeats itself six times; then there is one entry for the number of bins per scan (200 in the above example), and finally the last entry is the number of pitch and roll entries per scan (20 in the above example).

The first entry of the command file tells the instrument to record 38 scans. The first scan will be a "blocked beam" scan. The calibration sphere is rotated into the place for the first scan of each flight line (command file entry). Data from the "dark" scan is used to calculate the RMS noise which in turn is used to calculate the system signal-to-noise ratio. The second scan should be discarded as the calibration sphere is being rotated out of the optical path during this scan. Therefore only 36 of 38 scans will contain valid spectral data.

The PC Real Time Clock is used as the source of time to determine when to begin a flight line. Please note that the PC Real Time Clock time is different than the PC tick time which DOS normally uses. The PC tick time interrupt is used by the flight software to determine when, during a scan, to sample the spectral data and so in no way corresponds to the PC Real Time Clock time. Normally when a DOS machine boots, the BIOS reads the (battery backed up) Real Time Clock time and converts this time into ticks since midnight (one tick is equal to 54.9 milliseconds). This value is loaded into memory location 0:46C. The 8253 timer interrupts the CPU at the 54.9 millisecond rate and the interrupt service routine increments the value stored at location 0:46C. The DOS "TIME" command reads this memory location when it displays or sets the time. The NASIC flight software commandeers this interrupt service routine and changes the interrupt rate of the 8253 so that DOS time becomes useless. The NASIC software always reads the PC Real Time Clock whenever "real" time is needed. What all this means is that when setting the system time on the NASIC instrument using the DOS "TIME" command, make sure that the NASIC flight code has not run on the system since the last boot. A utility should be added to the flight software to restore the proper number of timer ticks into memory location 0:46C before returning to DOS.

4.2 Data Format

Data taken by NASIC is written to the solid state IDE hard drive into a file called "nasX.dat" where X is the number of the flight line. In other words, data from the first flight line is written to a file called nas0.dat, the second flight line data is written to the file "nas1.dat" and so on up to "nas5.dat" if there are a total of six flight lines.

Data is written to the file in 2K byte binary blocks. One block is assembled for each five second scan of the instrument. The number of spectral bins per scan is programmable but the nominal value is 200 bins per scan. Given the 200 bins/scan value, each block will contain 200 low gain spectral data points, 200 high gain spectral data points, 1 data point from each of the other six MUX inputs, and real time.

Each block is assembled as a linked list. The first integer of each block identifies a data type. The second integer of each block identifies the length in integers of that data type. The beginning of the next data type can be found by adding the length of the current data type to the address of the current data type label. A program called "dcode.c", included in the appendix, is a utility which demonstrates how this is done.

Type of Data	Type Identifier	Length Identifier
MUX Channel 0 (Spare)	0x0000	0x000f
Ground	0x0001	0x000f
Zener Diode (6.95 V)	0x0002	0x000f
Frame Temperature	0x0003	0x000f
Negative Supply	0x0004	0x000f
Detector Temperature	0x0007	0x000f
Real Time	0x0010	0x000f
LowGainSpectral (No Filter)	0x0020	BinCount
LowGainSpectral (Filter)	0x0021	BinCount
HighGainSpectral (No Filter)	0x0030	BinCount
HighGainSpectral (Filter)	0x0031	BinCount
LowGainDarkData	0x0040	BinCount
HighGainDarkData	0x0041	BinCount
End of Data	0x0069	0

There are two different versions of the flight software that affect the data format. The NASIC Principal Investigator has not made a decision as to which version to use. The first version has a blue filter drop into the optical path just behind the telescope on alternate scans. In this scenario, one 2K block would have data types 0x0020 and 0x0021 and the next would have data types 0x0030 and 0x0031. The purpose of the filter is to prevent second order effects from the blue end of the spectrum from contaminating the red end of the spectrum. Therefore the filter is really only needed at wavelengths longer than 700 nm. In the alternate software version the filter is switched into the optical path midway through each scan. In this case, data types 0x0020 and 0x0030 are identical (both being low gain data) and data types 0x0021 and 0x0031 are identical (both being high gain data).

This first version where the filter drops into the optical path on alternate scans requires that half of all data be trashed before analysis. (Only half the spectral data of any given scan would be valid.) This was the way the old system worked. The second version results in the acquisition of nearly twice as much data and simplifies data analysis but has the disadvantage of losing two or three spectral bins in the middle of each scan. This may or may not be a problem depending on the goals of a particular flight mission.

Included in the appendix is the source code for a program called "dcode.c". This program serves as an example of how to decode the flight data.

4.3 Flight Software Operation

When the pilot applies power to NASIC, the PC compatible computer boots and executes the "autoexec.bat" file. The last entry in the autoexec should be the name of the executable flight code. When the instrument was shipped to Greenbelt the executable was called "run". The first order of business for the flight software is to read the command file called "nas.cmd". This text file should be in the same directory as the "run" program. After successfully reading the command file the system will wait until the computer real time clock matches the first time entry in the command file. At this time the turret will be moved to the proper azimuth and elevation, and the stepper motor that drives the grating will be initialized and set into motion. The instrument will now rotate the calibration sphere into the optical path and take a dark scan. The dark scan data will later be used to determine the system electronic noise. After the dark scan data has been acquired and written, the calibration sphere is rotated out of the optical path and the programmed number of data scans are recorded.

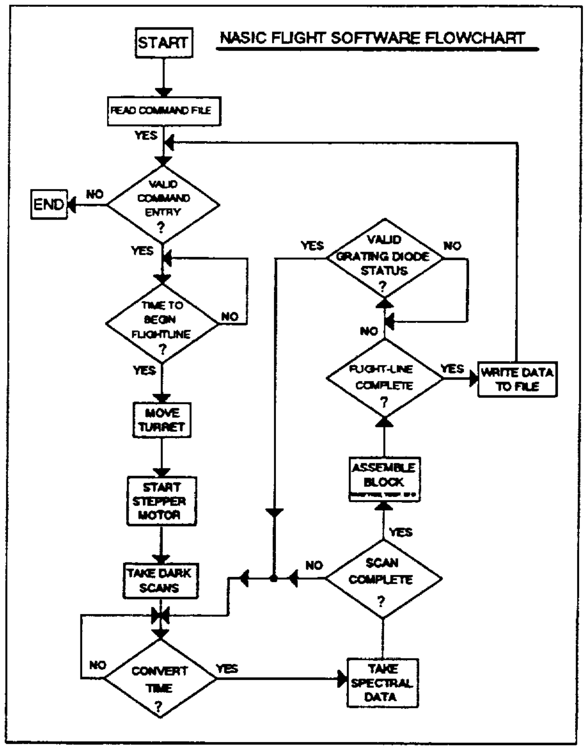


Figure 3. Flight Software

Each scan is made up of a programmable number of spectral bins. Normally this will be set to 200 bins. It is known that it takes the stepper motor 4.9 seconds to complete a scan. The beginning and end of scan are marked by polling an optical position sensor which detects a notch in a disk coupled to the stepper motor. The flight code divides 4.9 seconds by the number of bins per scan to determine the time interval between each spectral bin. This value is loaded into registers within the 8254 programmable interval timer. The Terminal Count output of the 8254 Timer is tied to the IRQ0 line of the computer. This is the highest priority interrupt so that there is virtually no chance of there being any significant skew in the triggering of this interrupt. (Only a block mode DMA transfer could supersede an interrupt and there are no block mode DMA transfers performed during acquisition time.) The interrupt service routine sets a flag which is monitored in a tight polling loop. Once the flag is set, the photo-diode channels are sampled and the interrupt flag is reset. Once the programmed number of bins are sampled, the program waits until the position sensor determines the beginning of the next scan. There is a dead time during which the grating returns to its home position. During this interval time, temperature and other ancillary data are taken and recorded.

Once the flight lines have been completed, i.e. the programmed number of scans have been completed, the entire set of data from the flight line is transferred from memory buffers on the PC motherboard to the solid state hard drive. The program then waits until it is time to begin the next flight line or, if the command file indicates that all flight lines are complete, then the program terminates.

4.4 Software Enhancements

Listed below is a summary of software enhancements made to the instrument.

Spectral Bins - The old system had a fixed number of spectral bins per scan. The new system allows one to set the number of spectral bins per scan in the command file.

Number of Scans per Flight Line - The old system was set so only 20 scans per flight line were allowed. The new system has a limit of about 150 scans per flight line assuming 200 bins per scan. The size of the buffers can be changed if necessary to make this much larger.

Real Time Averaging - The old system took one sample per spectral bin and recorded it. The new system can take up to 8 samples per spectral bin and average them before recording.

Operation of Blue Filter - The old system switched in the filter on alternate scans thereby wasting half of all data collected. The new system provides for the option of switching in the filter mid-way through a scan.

Diagnostics - The old system had no real time diagnostic facility. The new system has the program "nasic.exe" to quickly check most system functions.

5.0 DATA ANALYSIS

A calibration sphere traceable to NIST was used as a source in the lab to perform some basic calibration and performance analysis of the instrument. All analysis was done on a PC using Matlab, a math package published by The Math Works, Inc..

5.1 Wavelength Registration

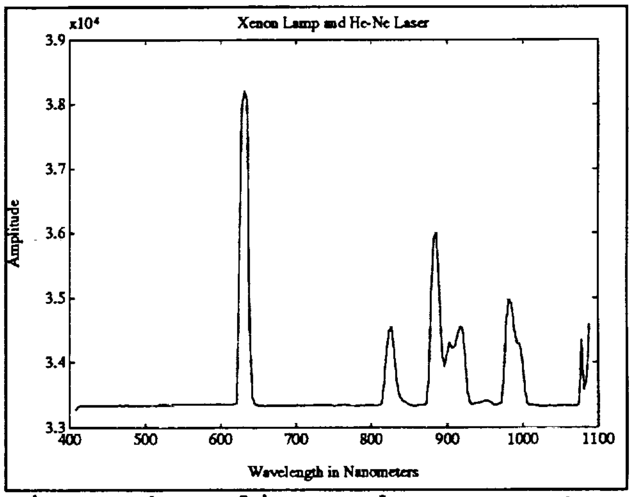


Figure 4. Helium and Xenon Data

The first order of business after the instrument became operational was to assign a wavelength to each data bin. We did this by using a helium-neon laser, a xenon lamp, and a mercury lamp. First we used the laser and the xenon lamp simultaneously as a source. The He-Ne line is known to be at 632.8 nm. Using this as a starting point we were able to identify the xenon peaks from a handbook containing spectral emission data. The same routine was repeated using a mercury source and the He-Ne laser. Figures 4 and 5 show the results of using these sources. The amplitude of all the plots is in terms of A/D counts.

5.2 Signal to Noise Analysis

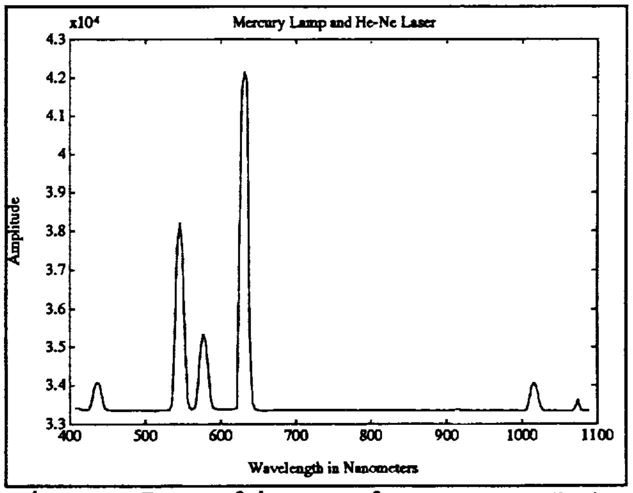


Figure 5. Helium and Mercury Data

After determining our wavelength axis we next set out to determine the performance of the instrument in terms of signal to noise. There is a calibration sphere on the instrument which can be rotated in front of the telescope to cut out all light. This "dark scan" data shows us the noise floor of the instrument. Figure 6 shows a plot of the high and low gain scans of dark data. Notice that there is a high degree of correlation between the high and low gain noise. This indicates that the system is not quantizer limited; in other words, the A/D is accurately measuring noise which appears at its input. If there is little or no correlation between the high and low gain dark scans then one could reasonably say that the system is quantizer limited or that the system noise is lower than the resolution of the A/D.

The standard deviation of the low gain dark scan data is usually around 8 counts whereas the standard deviation of the high gain dark data is about 16 counts. The high gain channel has twice the gain of the low gain channel, so the fact that the standard deviation of one channel is twice that of the other is reassuring and is yet another indication that the system is not quantizer limited.

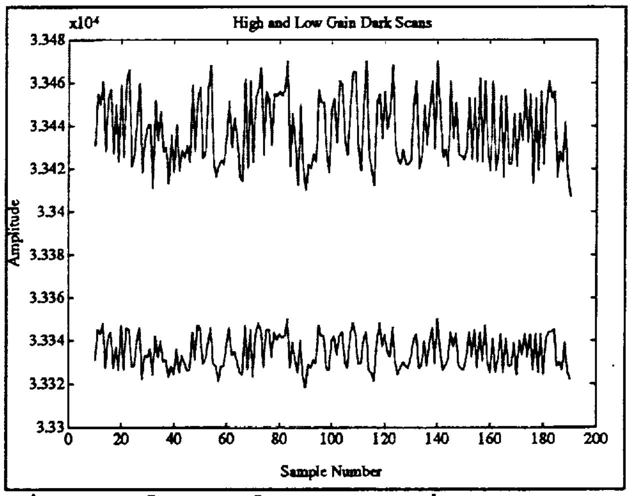


Figure 6. Dark Scan Noise Data

The system as it was before 1992 yielded a dark scan standard deviation of about 1 count on the low gain channel. That system used a 12-bit quantizer so that the standard deviation for comparison's sake would have to be multiplied by 16 (in order to compare noise from a 12-bit quantizer versus a 16-bit quantizer). After the adjustment, the old system has a low gain dark scan noise standard deviation of about 16 counts where the current system has a low gain dark scan standard deviation of about 8 counts. We therefore conclude that system noise in the new NASIC has been reduced by a factor of two.

$$SNR=\frac{2^{(n-1)}}{\sqrt{\left(\frac{1}{\sqrt{12}}\right)^2+\sigma^2}} \quad \text{Eq 1.} \; n=\text{bits} \quad \sigma=\text{RMS noise}$$

All signal to noise (SNR) calculations are made using the calibration sphere with one lamp. The SNR is first calculated for a single scan and then computed again for a set of averaged scans. Equation 1 shows how we calculate signal to noise.

Using equation 1 and plugging in the values of noise and number of quantization levels from the old system and new, we find that the maximum possible SNR of the new system is 2570 (about 11.5 bits) and the best possible SNR of the old system was 1245 (about 10 bits). This assumes that the full dynamic range of the A/D is being used.

Ideally one would expect the SNR to increase as the number of scans averaged together increased. Unfortunately, that assumes that one has a stable wavelength registration. In this instrument the chain drive on the diffraction grating causes a jitter which degrades one's ability to accurately determine wavelength, and this translates into a diminution in SNR when multiple scans are averaged. Figure 7 shows a typical waveform generated by plotting one scan of data obtained by viewing the calibration sphere. The sharp drop at 700 nm is the location where the blue filter is switched in. As one expects, the SNR curve is exactly the same as the actual sphere data curve. Figure 8 shows a single scan SNR curve and the SNR curve when 10 scans are averaged. The multiple scan SNR is calculated by using the averaged waveform as the signal and calculating the point by point standard deviation of all 10 waveforms and then plugging these values into Equation 1. A visual inspection of the two curves indicates that the multiple scan SNR is related to the inverse of the derivative of the single scan SNR curve. Notice that where the slope of the spectral data is greatest, the the SNR is lowest. This relationship is exactly what one predicts would happen if there is jitter in the motion of the diffraction grating.

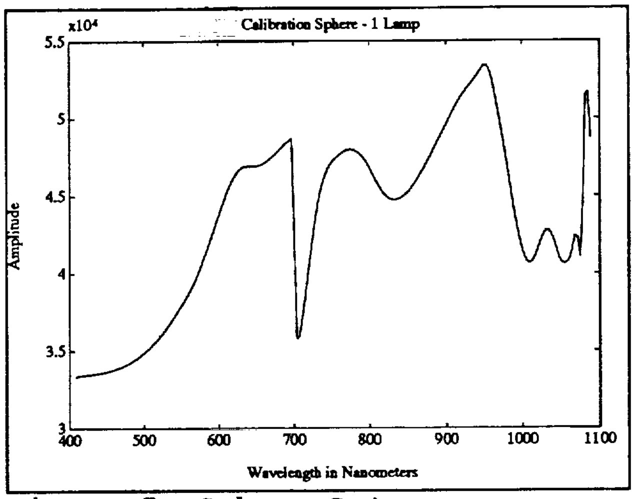


Figure 7. Sphere Data

These curves indicate that it is very problematic to specify the SNR performance of this instrument. The SNR will always depend on the spectral profile of the source. It may be possible to fix the data in software by deducing the effect of the jitter and then removing it, but the algorithms to do this might be more costly to develop than simply reworking the drive mechanism of the instrument to eliminate the jitter.

6.0 CONCLUSIONS

The primary purpose of reworking the instrument was to bring the technology of the system up to date and in so doing hopefully increase the perfomance of the instrument to such a degree that it could be used to calibrate SeaWifs and MODIS.

New and up to date technology has been applied to the data system and control functions of the instrument in such a way as to increase reliability, maintainability, and flexibility.

Unfortunately, the quality of the final data product has not been improved at all due to the mechanical problems associated with the diffraction grating.

The jitter of the grating in the NASIC instrument causes such a severe degradation in the SNR that, unless extensive software development is undertaken to correct the data post flight, serious consideration should be given to reworking the diffraction grating drive mechanism.

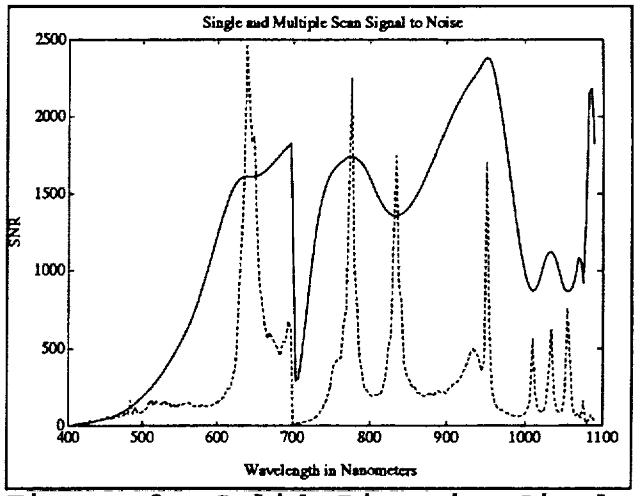


Figure 8. Solid Line is Single Scan SNR. Dashed Line is Multiple Scan SNR.

REFERENCES

1. Abel, P., Galimore, R., and Cooper, J., "Calibration results for NOAA-11 AVHRR Channels 1 and 2 from congruent aircraft data", in internal review (1992b).

2. Smith, G.R., Levin, R.H., Abel, P., and Jacobowitz, H., "Calibration of the solar channels of the NOAA-9 AVHRR using hihg altitude aircraft measurements" J.Atm and Ocean Tech.,5: p. 631-639, (1988).

3. Che, N., Grant, B.G., Flittner, D.E., Slater, P.N. Biggar, S.F. "Results of calibrations of the NOAA-11 AVHRR made by reference to calibrated SPOT imagery at White Sands, N.M., SPIE, 1493: p. 182-194 (1991).

4. Abel, P., "Clouds as calibration targets for AVHRR reflected-solar channels - Results from a two-year study at NOAA/NESDIS", Calibration of passive remote observing optical and microwave instrumentation; Proceedings from the Meeting, Orlando, Fl. Apr. 3-5, 1991. Society of Photo-Optical Instrumentation Engineers, p. 195-206, (1991).

[illegible] 16 [illegible]

8. APPENDIX A - NASIC SOFTWARE LISTINGS

```
/* This program decodes the NASIC flight data  */


#include<stdio.h>
#include<io.h>
#include<fcntl.h>
#include<sys\stat.h>
#include<alloc.h>

unsigned int   i,j,k,TypeCount,TargetType,SkipFirst;

unsigned char c,d,e,done,ScanCount=0;
unsigned char *cr;
FILE *fp,*fp1,*fp2,*fp3,*fp4,*fp5,*fp6,*fp7;
unsigned int *buffer,*buff,*BufferBase;
unsigned int DataType,DataLength;
struct stat FileInfo;
unsigned long FileSize;
char DataFileName[20],StringName[80];
char ValidDataFilesAvailable=1,True=1,False=0;
float AvgDiode[255];
float diode [8] [255];
void main(){

      c=0x2f;
      *cr=0x0d;
      buffer=malloc(0x800);  /* bufffer for standard 2K data block
*/
      BufferBase=buffer;      /* initialize pointers  */
      while(ValidDataFilesAvailable){
       ScanCount=0;
       c++;
       sprintf(DataFileName,"nas%c.dat",c);
       fclose(fp);
       fp=fopen(DataFileName,"rb");
       if(fp==NULL){
           printf("\n error opening nas%c.dat",c);
           ValidDataFilesAvailable=False;
           exit(0);
       }
       sprintf(DataFileName,"diode%c.20",c);
       fclose(fp1);
       fp1=fopen(DataFileName,"wb");
       if(fp1==NULL) printf("\n error opening file");
       sprintf(DataFileName,"diode%c.21",c);
       fclose(fp2);
       fp2=fopen(DataFileName,"wb");
       sprintf(DataFileName,"diode%c.30",c);
       fclose(fp3);
       fp3=fopen(DataFileName,"wb");
```

PRECEDING PAGE BLANK NOT FILMED

```
        sprintf(DataFileName,"diode%c.31",c);
        fclose(fp4);
        fp4=fopen(DataFileName,"wb");
        sprintf(DataFileName,"LDark%c.dat",c);
        fclose(fp5);
        fp5=fopen(DataFileName,"wb");
        sprintf(DataFileName,"HDark%c.dat",c);
        fclose(fp6);
        fp6=fopen(DataFileName,"wb");
        sprintf(DataFileName,"misc%c.dat",c);
        fclose(fp7);
        fp7=fopen(DataFileName,"wb");

        fstat(fileno(fp),&FileInfo);
        FileSize=FileInfo.st_size;          /*  FileSize in Bytes
*/
                                  /*  printf("\n  initial  buffer  is
%x",buffer); */

        while((float)ScanCount*0x800<(float)FileSize){
           printf("\nFile   Size   is   %6.0f   and   Pointer   is
%6.0f",(float)FileSize,(float)ScanCount*0x800);
           fread(buffer,sizeof(int),0x400,fp);
           DataType=0;
           while(((buffer-BufferBase)<0x800)&(DataType!=0x69)){
           DataType=*buffer;
           DataLength=*(++buffer);
           printf("\n   Data   Type   is   %x   and   Length   is
%x",DataType,DataLength);
           switch(DataType){
             case 0x20:  WriteLowGainNoFilterData(); break;
             case 0x21:  WriteLowGainWithFilerData(); break;
             case 0x30:  WriteHighGainNoFilterData(); break;
             case 0x31:  WriteHighGainWithFilterData(); break;
             case 0x40:  WriteLowGainDarkData(); break;
             case 0x41:  WriteHighGainDarkData(); break;
             case 0x00:  WriteChan0Data(); break;
             case 0x01:  WriteChan1Data(); break;
             case 0x02:  WriteChan2Data(); break;
             case 0x03:  WriteChan3Data(); break;
             case 0x04:  WriteChan4Data(); break;
             case 0x07:  WriteChan7Data(); break;
             case 0x10:  WriteType10Data(); break;
             case 0x69:  break;
             default:    printf("\n probable error"); break;
           }
           buffer+=DataLength+1;
           if(DataType==0x69) buffer=BufferBase;
           }
           ScanCount++;
        }
      }
```

```
}
WriteLowGainNoFilterData(){
      buff=buffer;
      buff++;
      for(i=0;i<DataLength;i++){
        fprintf(fp1,"%f\n",(float)*buff++);
      }
      fprintf(fp1,"\n");
      return(0);
}
WriteLowGainWithFilerData(){
      buff=buffer;
      buff++;
      for(i=0;i<DataLength;i++){
        fprintf(fp2,"%f\n",(float)*buff++);
      }
      fprintf(fp1,"\n");
      return(0);
}
WriteHighGainNoFilterData(){
      buff=buffer;
      buff++;
      for(i=0;i<DataLength;i++){
        fprintf(fp3,"%f\n",(float)*buff++);
      }
      fprintf(fp1,"\n");
      return(0);
}
WriteHighGainWithFilterData(){
      buff=buffer;
      buff++;
      for(i=0;i<DataLength;i++){
        fprintf(fp4,"%f\n",(float)*buff++);
      }
      fprintf(fp1,"\n");
      return(0);
}
WriteLowGainDarkData(){
      buff=buffer;
      buff++;
      for(i=0;i<DataLength;i++){
        fprintf(fp5,"%f\n",(float)*buff++);
      }
      fprintf(fp1,"\n");
      return(0);
}
WriteHighGainDarkData(){
      buff=buffer;
      buff++;
      for(i=0;i<DataLength;i++){
        fprintf(fp6,"%f\n",(float)*buff++);
      }
```

```
     fprintf(fp1,"\n");
     return(0);
}
WriteChan0Data(){
     buff=buffer;
     buff++;
     fprintf(fp7,"%s","\nChannel 0:");
     for(i=0;i<DataLength;i++){
       fprintf(fp7,"\n%f",(float)*buff++);
     }
     return(0);
}
WriteChan1Data(){
     buff=buffer;
     buff++;
     fprintf(fp7,"%s","\nChannel 1:");
     for(i=0;i<DataLength;i++){
       fprintf(fp7,"\n%f",(float)*buff++);
     }
     return(0);
}
WriteChan2Data(){
     buff=buffer;
     buff++;
     fprintf(fp7,"%s","\nChannel 2:");
     for(i=0;i<DataLength;i++){
       fprintf(fp7,"\n%f",(float)*buff++);
     }
     return(0);
}
WriteChan3Data(){
     buff=buffer;
     buff++;
     fprintf(fp7,"%s","\nChannel 3:");
     for(i=0;i<DataLength;i++){
       fprintf(fp7,"\n%f",(float)*buff++);
     }
     return(0);
}
WriteChan4Data(){
     buff=buffer;
     buff++;
     fprintf(fp7,"%s","\nChannel 4:");
     for(i=0;i<DataLength;i++){
       fprintf(fp7,"\n%f",(float)*buff++);
     }
     return(0);
}
WriteChan7Data(){
     buff=buffer;
     buff++;
     fprintf(fp7,"%s","\nChannel 7:");
     for(i=0;i<DataLength;i++){
       fprintf(fp7,"\n%f",(float)*buff++);
     }
     return(0);
}
WriteType10Data(){
     buff=buffer;
     buff++;
     fprintf(fp7,"%s","\nData Type 10, Time:");
     for(i=0;i<DataLength;i++){
       fprintf(fp7,"\n%x",*buff++);
     }
     return(0);
}


/*     This is the flight code for the NASIC instrument
*/
/*                                                */
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<dos.h>
#include<math.h>
#include<mem.h>

#define HeaderDataLength 56   /* Remember to change this if adding
data  */

unsigned char Filter,FlightLine;

unsigned int *LGNoFilter,*LGNoFilterBase,LGNoFilterIndex;
unsigned int *HGNoFilter,*HGNoFilterBase,HGNoFilterIndex;
unsigned int *LGWithFilter,*LGWithFilterBase,LGWithFilterIndex;
unsigned int *HGWithFilter,*HGWithFilterBase,HGWithFilterIndex;
unsigned int *MiscData,*MiscDataBase,MiscDataIndex,MiscScanIndex;
unsigned int LowGainNoFilter[512];
unsigned int LowGainWithFilter[512];
unsigned int HighGainNoFilter[512];
unsigned int HighGainWithFilter[512];

unsigned int HighGainDarkData[512];
unsigned int LowGainDarkData[512];
unsigned char DarkDataFlag;

float InterruptRate,TimerValue,ScanTime=4.9,freq=1193180.0;
unsigned TimerIntValue;
unsigned char TimerLow,TimerHigh;
char LastGratingDiodeStatus;
char CurrentGratingDiodeStatus;
char BeginScanTime;
unsigned int DetectorTemp,FrameTemp,LowGain,HighGain;
unsigned char LowGainLowByte,LowGainHighByte;
unsigned char HighGainLowByte,HighGainHighByte;
unsigned char SecondsRegister=0,ClockSeconds;
unsigned char MinutesRegister=2,ClockMinutes;
unsigned char HoursRegister=4,ClockHours;
unsigned long temp[8],htemp[8],ltemp[8];

unsigned char SphereOpenFlag,CurrentSphereStatus,PrevSphereStatus;

float     *fptr,fvalue;
unsigned int NumOfBins;
unsigned long SystemSeconds,ActivateSeconds;
unsigned int i,j,k,l,DataByteCnt;
unsigned int SpectralDataByteCount,*SpectralDataLengthPtr;
unsigned char c,d,e;
unsigned char ConvertTime,ScanCount,FileCount;
unsigned int FinalBinCount,BinCount;
unsigned int FinalIntCount,IntCount;
char *DataFileNamePtr;
char DataFileName[80];
char MiscFileName[80];

unsigned char CmdEntryCount,CurrentCmdActive;
FILE *fp1,*fp2,*fp3;
void interrupt NasIsr();   /* main interrupt service routine  */
void interrupt (*rest)(); /* vector to restore original routine */
struct command{   /* this structure is 15 bytes long  */
       unsigned char hours;          /* Hours to start Scan    */
       unsigned char minutes;           /* Minutes to start Scan
*/
       unsigned char seconds;           /* Seconds to start Scan
*/
       float azimuth;                   /* Azimuth in Degrees
*/
       float elevation;                 /* Elevation in Degrees
*/
       unsigned char NumOfScans;       /* Number of Scans     */
       unsigned  int  BinsPerScan;         /*  Number  of  Spectral
Bins/Scan      */
       unsigned char PitchRollPerScan; /* Num of Pitch/Roll Samples
per Scan*/
} CmdEntry[6];

void ReadCmdFile(){
     fptr=&fvalue;  /* this forces compiler to link floating point
formats */
     fp1=fopen("nasic.cmd","r");
     if(fp1==NULL) printf("\n Command File is not available");
     for(i=0;i<6;i++){
     fscanf(fp1,"%d%c",&CmdEntry[i].hours,&c);
     fscanf(fp1,"%d%c",&CmdEntry[i].minutes,&c);
     fscanf(fp1,"%d",&CmdEntry[i].seconds);
     fscanf(fp1,"%f",&CmdEntry[i].azimuth);
     fscanf(fp1,"%f",&CmdEntry[i].elevation);
     fscanf(fp1,"%d",&CmdEntry[i].NumOfScans);
     }
     fscanf(fp1,"%d",&CmdEntry[0].BinsPerScan);
/*     fscanf(fp1,"%d",&CmdEntry[0].PitchRollPerScan);  */
     for(j=1;j<6;j++){
     CmdEntry[i].BinsPerScan=CmdEntry[0].BinsPerScan;
/*   CmdEntry[i].PitchRollPerScan=CmdEntry[0].PitchRollPerScan; */
     }

     NumOfBins=(unsigned)CmdEntry[0].BinsPerScan;

}

void WaitTillProperTime(){

ActivateSeconds=(long)CmdEntry[CmdEntryCount].hours*3600+(long)Cm
dEntry[CmdEntryCount].minutes*60+(long)CmdEntry[CmdEntryCount].se
conds;
  printf("\n CmdEntryCount is %d",CmdEntryCount);
  printf("\n hours are  %d\n",CmdEntry[CmdEntryCount].hours);
  while((ActivateSeconds!=SystemSeconds)&!kbhit()){
      outportb(0x70,SecondsRegister);
      ClockSeconds=inportb(0x71);
      outportb(0x70,MinutesRegister);
      ClockMinutes=inportb(0x71);
      outportb(0x70,HoursRegister);
      ClockHours=inportb(0x71);
      SystemSeconds=(ClockHours>>4)*36000+(ClockHours&0x0f)*3600;

SystemSeconds=SystemSeconds+(ClockMinutes>>4)*600+(ClockMinutes&0
x0f)*60;

SystemSeconds=SystemSeconds+(ClockSeconds>>4)*10+(ClockSeconds&0x
0f);
      printf("\r %lu %lu",ActivateSeconds,SystemSeconds);
      delay(500);
      printf("\r        ");
    }

}
void MoveGimbal(){
#define AzCW  0x08
#define AzCCW 0x04
#define ElCW  0x01
#define ElCCW 0x02
#define MaxAzCnt 2510;
#define MaxElCnt 1354;
unsigned int   count=0;
int  AzHitCnt=0,ElHitCnt=0,FinalAzEncoderCnt=0,FinalElEncoderCnt=0;
char      AzHit,ElHit,match=0,sw,quit=0,CalStatus=0;
```

```
/*****        The first task is to Calibrate the Gimbal
********/
/***************************************************************
******/
     outportb(0x109,0); /* this turns off all motors,solenoids
etc.*/
     delay(100);
     outportb(0x109,AzCW+ElCW);    /* Swing the Gimbal Azimuth
Clockwise  */
     delay(1000);      /* Raise the Gimbal Elevation Clkwise  */
     while(!quit){
        AzHit=inportb(0x10a);   /*read Azimuth shaft angle encoder
*/
        AzHit=(AzHit>>7)&1;      /* mask out all other bits
*/
/*      printf("\n AzHit=%x match=%x",AzHit,match); /*  */
        if(AzHit==match){        /* has encoder state changed?
*/
           AzHitCnt++;
           count=0;              /* reset the counter
*/
           match=(AzHitCnt+2)%2; /* invert match
*/

        }
        delay(1);
        count++;
        if(count==1000) quit=1; /* if Gimbal is on stops we'll know
*/
     }
     quit=0;
     count=0;
     while(!quit){
        ElHit=inportb(0x10a);
        ElHit=(ElHit>>3)&1;
        if(ElHit==match){
           ElHitCnt++;
           count=0;
           match=(ElHitCnt+2)%2;
        }
        delay(1);
        count++;
        if(count==1000) quit=1;
     }
     outportb(0x109,0);
     AzHitCnt=ElHitCnt=0;


     printf("\n the gimbal is now calibrated ");

/******************  The Gimbal is now Calibrated ************/
/*************************************************************/
```

```
/******************  Now Move Gimbal to Proper Azimuth *******/
/****************************************************************/

FinalAzEncoderCnt=(int)(CmdEntry[CmdEntryCount].azimuth/200.0*250
9.0);
   printf("\n AzEncoder Count is %d\n",FinalAzEncoderCnt);
   if(AzHitCnt<FinalAzEncoderCnt) outportb(0x109,AzCCW);
   else outportb(0x109,AzCW);
   while(AzHitCnt!=FinalAzEncoderCnt){
        AzHit=inportb(0x10a); /*read Azimuth shaft angle encoder */
        AzHit=(AzHit>>7)&1;   /* mask out all other bits         */
        if(AzHit==match){     /* has encoder state changed?      */
           count=0;           /* reset the counter               */
           if(AzHitCnt>FinalAzEncoderCnt) AzHitCnt--;
           else AzHitCnt++;
           match=(AzHitCnt+2)%2; /* invert match          */
        }
        delay(1);
        count++;
        if(count==100) quit=1; /* is Gimbal is on stops? */
     }  printf("\n the final az count is %d",AzHitCnt);
     outportb(0x109,0);

/**********  Gimbal Now at Proper Azimuth  (We Hope) ***********/
/****************************************************************/

/************ Now Move Gimbal to Proper Elevation  *************/
/***************************************************************/


FinalElEncoderCnt=(int)(CmdEntry[CmdEntryCount].elevation/50.0*13
54.0);

FinalElEncoderCnt=(int)(sin(CmdEntry[CmdEntryCount].elevation/2.0
/360.0*2*3.1415)*1354.0*1.9);
    FinalElEncoderCnt    =    (int)(1480.0    *    sqrt(1.84    *
(1-cos(CmdEntry[CmdEntryCount].elevation*.01745))));
    /* FinalElEncoderCnt+=75;  /* */
    printf("\n Final EL Encoder Count is %d",FinalElEncoderCnt);
    if(ElHitCnt<FinalElEncoderCnt) outportb(0x109,ElCCW);
    else outportb(0x109,ElCW);
    while(ElHitCnt!=FinalElEncoderCnt){
    ElHit=inportb(0x10a);   /*read Azimuth shaft angle encoder */
        ElHit=(ElHit>>3)&1;   /* mask out all other bits         */
        if(ElHit==match){     /* has encoder state changed?      */
           count=0;           /* reset the counter               */
           if(ElHitCnt>FinalElEncoderCnt) ElHitCnt--;
           else ElHitCnt++;
           match=(ElHitCnt+2)%2; /* invert match             */
        }
        delay(1);
        count++;
        if(count==100) quit=1; /* is Gimbal is on stops ? */
     }  printf("\nthe final el count is %d", ElHitCnt);
     outportb(0x109,0);

/*********  Gimbal is now at Proper Elevation  ******************/
}

void StartStepper(){

#define base    0x150
#define data    base+0
#define stat    base+1    /* when reading it is status  */
#define command base+1    /* when writing it is command */
#define aux     base+2

#define   stop 0x40

/********* Status Register Format *********************************/
/*                                                                */
/* Output Buffer Full - Bit 0.  must be 1 to read data            */
/* Input Buffer Full  - Bit 1.  must be 0 to write commands       */
/* Motor Busy         - Bit 2.  Flag is 1 during motor operation  */
/*                                                                */
/******************************************************************/

char busy;
void StepperStatus();
/*******************  Initialization ******************************/

  outportb(aux,0xff);   /* D7=1 will reset the card        */
  delay(10);          /* this delay is necessary         */
  outportb(aux,0x40);   /* D6=0 activates card, so not yet */
  StepperStatus();
  outportb(command,0x01); /*positive logic,3 phase,2 phase step */

  StepperStatus();
  outportb(data,0x60);          /* RA max */
  StepperStatus();
  outportb(data,0x20);         /*  RA min  */
  StepperStatus();
  outportb(data,0x40);         /* accel LSB  */
  StepperStatus();
  outportb(data,01);           /* accel MSB  */
  StepperStatus();
  outportb(aux,0);       /* activate phases */
  StepperStatus();

/*********************    End   of   Initialization
*************************/

/*********** Find Begin of Scan ******************************/
```

```
  outportb(command,0x4c);   /* constant speed operation command */
  StepperStatus();
  outportb(data,0x25);      /* 0x25 is the rate */
  StepperStatus();
  outportb(data,0xcf);        /* number of steps LSByte  */
  StepperStatus();
  outportb(data,7);           /*   middle byte  */
  StepperStatus();
  outportb(data,0xff);            /*  MSByte        */
}
void StepperStatus(){
char StepperByte;
  StepperByte=1;
  while(StepperByte){
    StepperByte=inportb(0x151);
    StepperByte=StepperByte&0x02;
  }
}
void FirstTimeInterruptInit(){
     disable();
     InterruptRate=(float)NumOfBins/ScanTime;
     TimerValue=freq/InterruptRate;
     TimerIntValue=(unsigned)TimerValue;
     TimerLow = (char)(TimerIntValue&0xff);
     TimerHigh = (char)((TimerIntValue>>8)&0xff);
     outportb(0x43,0x36);  /* set up the timer chip */
     outportb(0x40,TimerLow);
     outportb(0x40,TimerHigh);
     rest=getvect(0x1c);
     setvect(0x1c,NasIsr);
     outportb(0x21,0);
     disable();
}
void InitializeInterrupts(){
     disable();
     outportb(0x43,0x36);  /* set up the timer chip */
     outportb(0x40,TimerLow);
     outportb(0x40,TimerHigh);
     enable();
}
void interrupt NasIsr(){
     ConvertTime=1;

}


void OpenSphere(){
   SphereOpenFlag=0;
   CurrentSphereStatus=PrevSphereStatus=0;
   while(!SphereOpenFlag){
      outportb(0x109,0x20);  /* start the cal sphere motor  */
      PrevSphereStatus=CurrentSphereStatus;
```

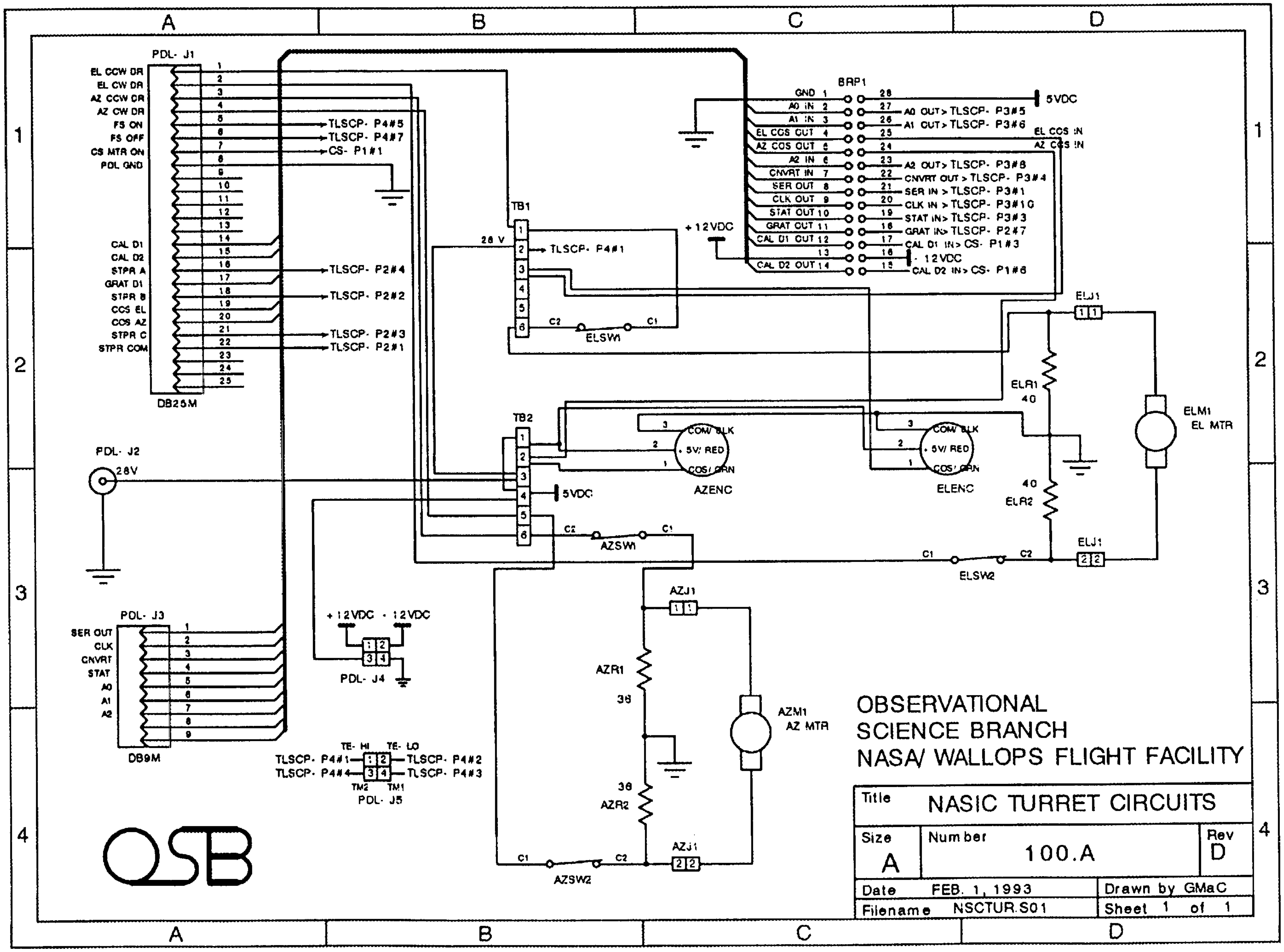
```
        CurrentSphereStatus=inportb(0x10a);
        CurrentSphereStatus=(CurrentSphereStatus&0x04)>>2;
        if((CurrentSphereStatus==1)&&(PrevSphereStatus==0)){
       SphereOpenFlag=1;
       outportb(0x109,0);
        }
    }
}

void CloseSphere(){

      outportb(0x109,0x20);
      delay(1500);
      outportb(0x109,0);
}

void StopStepper(){
       outportb(0x151,0x40);
}

void CleanUp(){
       disable();
       setvect(0x1c,rest);
       outportb(0x43,36);
       outportb(0x40,0xff);
       outportb(0x40,0xff);
       enable();
}

void AssembleBlock(){

 /* Channels 0,1,2,3,4,7 are to be collected and stored  & time */

      BinCount=0;
      outportb(0x109,0x0);   /* flip the filter out  */
      for(i=0;i<5;i++){
        outportb(0x108,0x08+i);
        delay(1);
        outportb(0x108,i);   /* high gain conversion  */
        delay(1);           /* wait till conversion is complete  */
        MiscDataIndex=MiscScanIndex+i*8;
        *(MiscData+MiscDataIndex)= (unsigned int)i;  /* data type  */
        MiscDataIndex++;
        *(MiscData+MiscDataIndex)= 6;              /* data length */
        MiscDataIndex++;
              *(MiscData+MiscDataIndex)     =
(255-inportb(0x108))+(255-(inportb(0x109)<<8));
      }
        outportb(0x108,0x08+7);
        delay(1);
        outportb(0x108,7);  /*  conversion  */
        delay(1);           /* wait till conversion is complete  */
      MiscDataIndex=MiscScanIndex+5*8;
      *(MiscData+MiscDataIndex++)=0x0007;   /* data type 7 for
channel 7 */
      *(MiscData+MiscDataIndex++)=6;       /* length */
           *(MiscData+MiscDataIndex++)=
(255-inportb(0x108))+(255-(inportb(0x109)<<8));
      *(MiscData+MiscDataIndex)=FinalIntCount;
      MiscDataIndex=MiscScanIndex+6*8;
      *(MiscData+MiscDataIndex++)=0x0010; /* Type Ten for Time  */
      *(MiscData+MiscDataIndex++)=6;
      outportb(0x70,HoursRegister);
      *(MiscData+MiscDataIndex++)=(unsigned int)inportb(0x71);
      outportb(0x70,MinutesRegister);
      *(MiscData+MiscDataIndex++)=(unsigned int)inportb(0x71);
      outportb(0x70,SecondsRegister);
      *(MiscData+MiscDataIndex++)=(unsigned int)inportb(0x71);
      MiscScanIndex=MiscScanIndex+7*8;
      if(!Filter){

movmem(HighGainNoFilter,HGNoFilter+HGNoFilterIndex,(unsigned)NumO
fBins*2);
         HGNoFilterIndex+=(unsigned)NumOfBins;

movmem(LowGainNoFilter,LGNoFilter+LGNoFilterIndex,(unsigned)NumOf
Bins*2);
         LGNoFilterIndex+=(unsigned)NumOfBins;
      } else{

movmem(HighGainWithFilter,HGWithFilter+HGWithFilterIndex,(unsigne
d)NumOfBins*2);
         HGWithFilterIndex+=(unsigned)NumOfBins;

movmem(LowGainWithFilter,LGWithFilter+LGWithFilterIndex,(unsigned
)NumOfBins*2);
         LGWithFilterIndex+=(unsigned)NumOfBins;
         }


}

void AssembleDarkBlock(){

 /* Channels 0,1,2,3,4,7 are to be collected and stored  & time */

    for(i=0;i<5;i++){
      outportb(0x108,0x08+i);
      delay(1);
      outportb(0x108,i);            /* high gain conversion  */
      delay(1);          /* wait till conversion is complete  */
      MiscDataIndex=MiscScanIndex+i*8;
      *(MiscData+MiscDataIndex++)= (unsigned int)i; /*data type */
      *(MiscData+MiscDataIndex++)= 6;           /* data length */
```

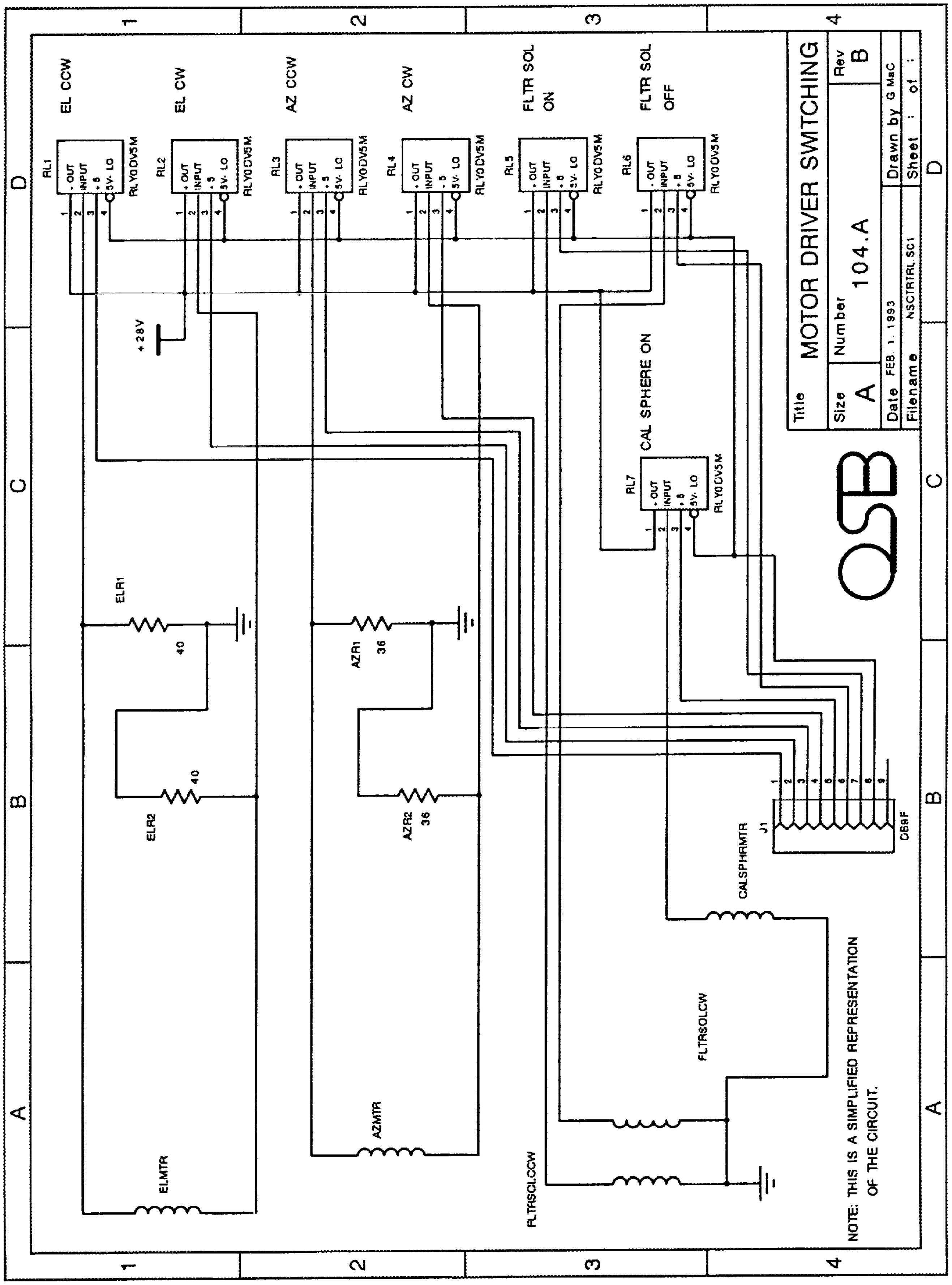
```
           *(MiscData+MiscDataIndex)    =
(255-inportb(0x108))+(255-(inportb(0x109)<<8));
    }
      outportb(0x108,0x08+7);
      delay(1);
      outportb(0x108,7);          /* high gain conversion  */
      delay(1);       /* wait till conversion is complete  */
      MiscDataIndex=MiscScanIndex+5*8;
      *(MiscData+MiscDataIndex++)=0x0007;/* type 7 for chan 7 */
      *(MiscData+MiscDataIndex++)=6;     /* length */
           *(MiscData+MiscDataIndex)=
(255-inportb(0x108))+(255-(inportb(0x109)<<8));
      MiscDataIndex=MiscScanIndex+ 6*8;
      *(MiscData+MiscDataIndex++)=0x0010; /* Type Ten for Time  */
      *(MiscData+MiscDataIndex++)=6;
      outportb(0x70,HoursRegister);
      *(MiscData+MiscDataIndex++)=(unsigned int)inportb(0x71);
      outportb(0x70,MinutesRegister);
      *(MiscData+MiscDataIndex++)=(unsigned int)inportb(0x71);
      outportb(0x70,SecondsRegister);
      *(MiscData+MiscDataIndex++)=(unsigned int)inportb(0x71);
      MiscScanIndex=MiscScanIndex+7*8;
}

void WriteToFile(){
      unsigned lgdtype=0x40;
      unsigned hgdtype=0x41;
      unsigned lgnftype=0x20;
      unsigned hgnftype=0x21;
      unsigned lgwftype=0x30;
      unsigned hgwftype=0x31;
      unsigned eodtype=0x69;
      unsigned bins=(unsigned)NumOfBins;
      unsigned zeros[2048];
      unsigned Scount=0;
      long CurrentPos;

     for(i=0;i<2048;i++) zeros[i]=0;
     CurrentPos=ftell(fp2);
     printf("\n the current position is %ld",CurrentPos);
     for(i=0;i<CmdEntry[(CmdEntryCount)].NumOfScans;i++){
       fwrite(zeros,sizeof(int),1024,fp2);
      }        /*            */

      fseek(fp2,0,SEEK_SET);      /*  */
      fwrite(MiscData,sizeof(int),HeaderDataLength,fp2);
      MiscData+=HeaderDataLength;
      fwrite(&lgdtype,sizeof(int),1,fp2);
      fwrite(&bins,sizeof(int),1,fp2);
      fwrite(LowGainDarkData,sizeof(int),(unsigned)NumOfBins,fp2);

      fwrite(&hgdtype,sizeof(int),1,fp2);
```

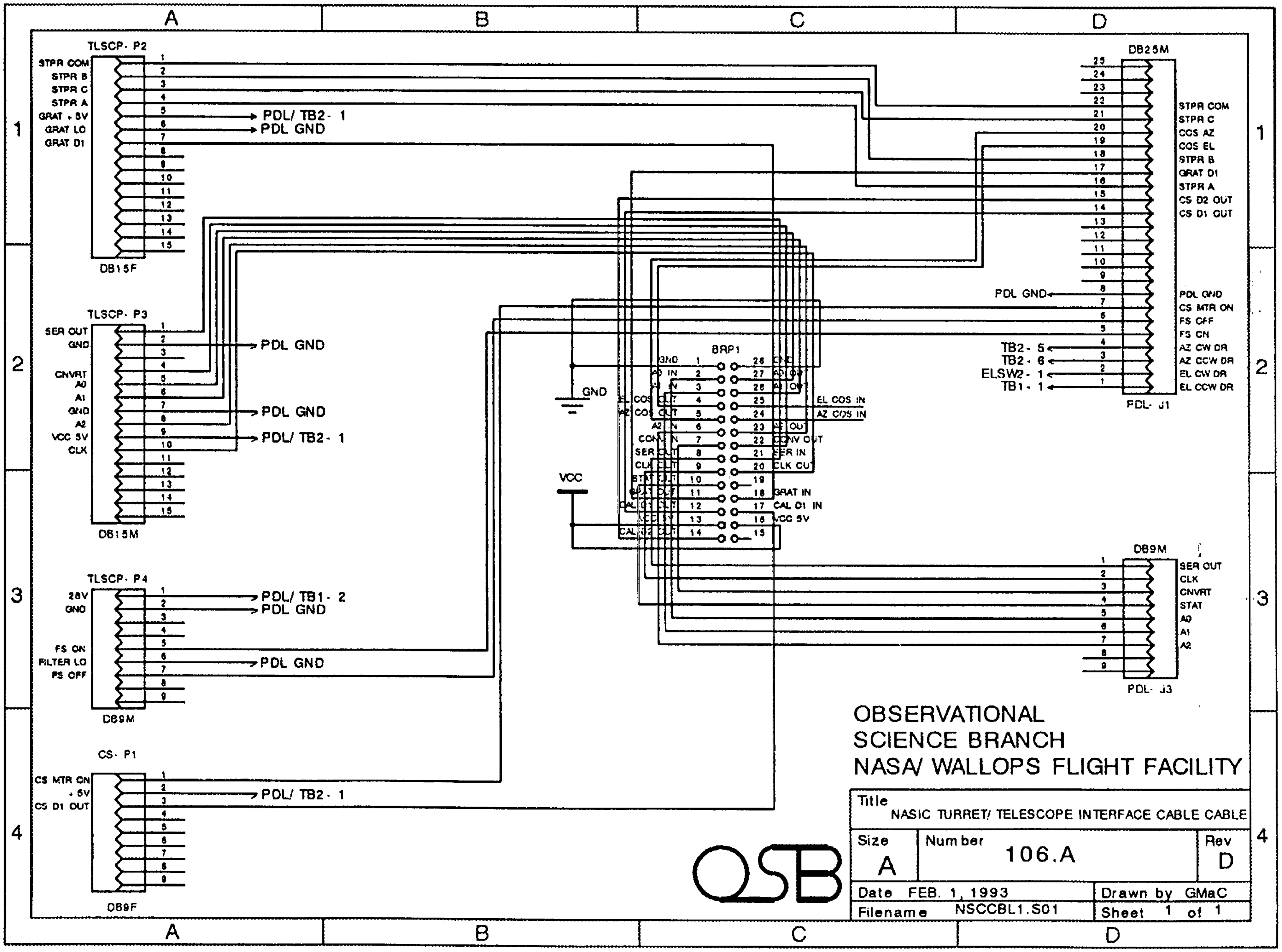
```
	fwrite(&bins,sizeof(int),1,fp2);
	fwrite(HighGainDarkData,sizeof(int),(unsigned)NumOfBins,fp2);
	fwrite(&eodtype,sizeof(int),1,fp2);
	for(i=0;i<CmdEntry[(CmdEntryCount)].NumOfScans;i++){
	fseek(fp2,(long)((i+1)*2048),SEEK_SET); /* */
	fwrite(MiscData,sizeof(int),HeaderDataLength,fp2);
	MiscData+=HeaderDataLength;
	if(Scount%2==0){
	  fwrite(&lgnftype,sizeof(int),1,fp2);
	  fwrite(&bins,sizeof(int),1,fp2);
	  fwrite(LGNoFilter,sizeof(int),(unsigned)NumOfBins,fp2);
	  LGNoFilter+=(unsigned)NumOfBins;
	  fwrite(&hgnftype,sizeof(int),1,fp2);
	  fwrite(&bins,sizeof(int),1,fp2);
	  fwrite(HGNoFilter,sizeof(int),(unsigned)NumOfBins,fp2);
	     fwrite(&eodtype,sizeof(int),1,fp2);
	  HGNoFilter+=(unsigned)NumOfBins;
	} else {


	    fwrite(&lgwftype,sizeof(int),1,fp2);
	    fwrite(&bins,sizeof(int),1,fp2);
	    fwrite(LGWithFilter,sizeof(int),(unsigned)NumOfBins,fp2);
	    LGWithFilter+=(unsigned)NumOfBins;
	    fwrite(&hgwftype,sizeof(int),1,fp2);
	    fwrite(&bins,sizeof(int),1,fp2);
	    fwrite(HGWithFilter,sizeof(int),(unsigned)NumOfBins,fp2);
	    HGWithFilter+=(unsigned)NumOfBins;
	       fwrite(&eodtype,sizeof(int),1,fp2);
	  }
	Scount++;
	}

}

void main(){

  outportb(0x109,0);  /* disable all motors, solenoids etc. */
  outportb(0x10a,0); /* enable latch to control motors, solenoids,
etc. */
  StopStepper();
  printf("\n At the beginning of main\n");

  LGNoFilter=LGNoFilterBase=malloc(0xff00);
  HGNoFilter=HGNoFilterBase=malloc(0xff00);
  LGWithFilter=LGWithFilterBase=malloc(0xff00);
  HGWithFilter=HGWithFilterBase=malloc(0xff00);
  MiscData=MiscDataBase=malloc(0xff00);
  ReadCmdFile();
  printf("\n have just read the Command File\n ");
  FirstTimeInterruptInit();
  enable();
```

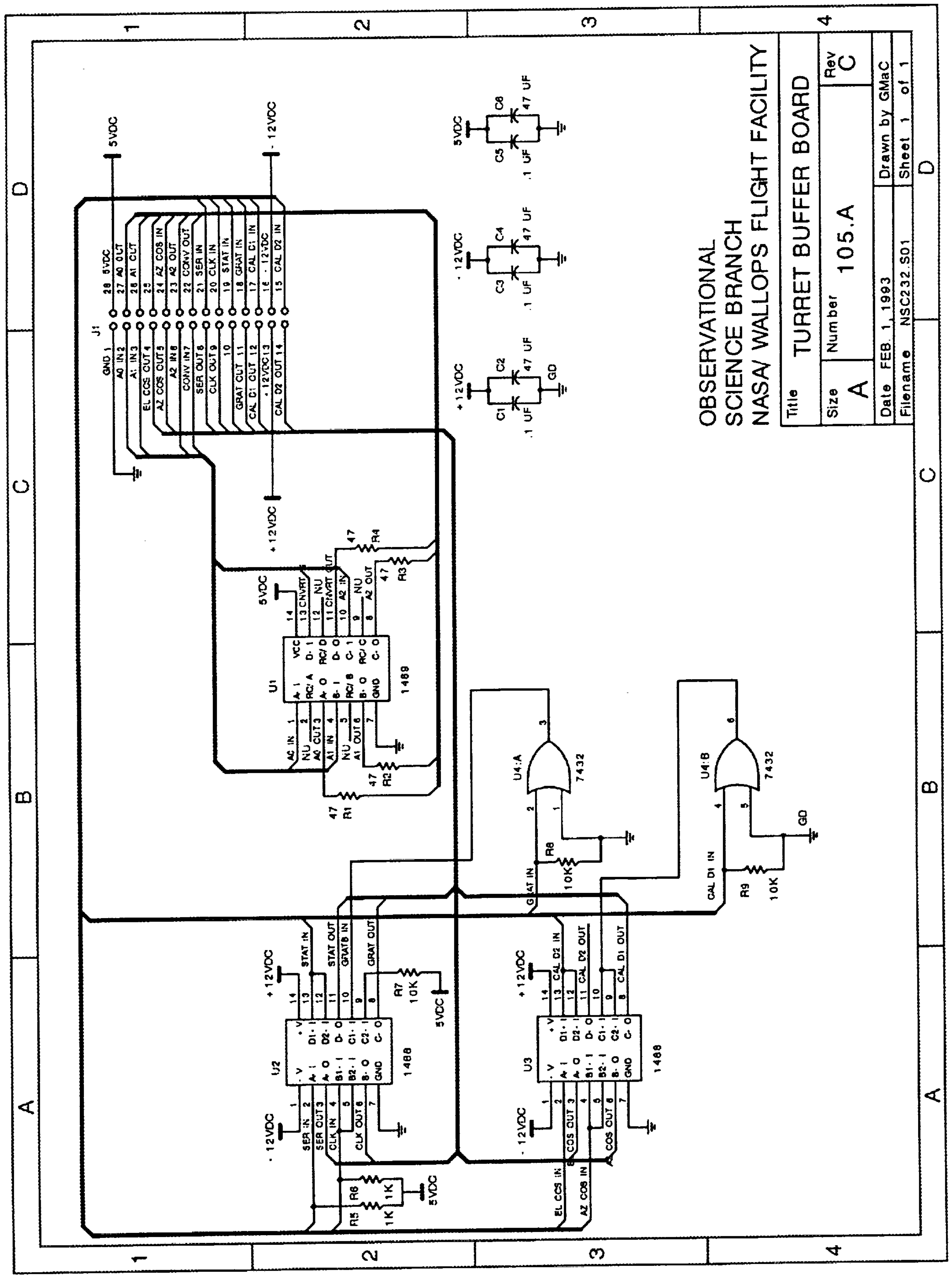
```
   for(FlightLine=0;FlightLine<6;FlightLine++){
      if((CmdEntry[CmdEntryCount].NumOfScans)!=0){
        DarkDataFlag=0;
        LGNoFilter=LGNoFilterBase;
        LGNoFilterIndex=0;
        HGNoFilter=HGNoFilterBase;
        HGNoFilterIndex=0;
        LGWithFilter=LGWithFilterBase;
        LGWithFilterIndex=0;
        HGWithFilter=HGWithFilterBase;
        HGWithFilterIndex=0;
        MiscData=MiscDataBase;
        MiscDataIndex=0; MiscScanIndex=0;
        for(i=0;i<0xff00/2;i++){
        *(LGNoFilter+i)=0;
        *(HGNoFilter+i)=0;
        *(LGWithFilter+i)=0;
        *(HGWithFilter+i)=0;
        *(MiscData+i)=0;
        }
        WaitTillProperTime();
        printf("\n just made it out of WaitTillProperTime\n");
        MoveGimbal();
        printf("\n have finished moving the gimbal into position");
        fclose(fp2);
        sprintf(DataFileName,"NAS%d.dat",FileCount++);
        fp2=fopen(DataFileName,"wb");
        if(fp2==NULL) printf("\n could not open the data file");
        StartStepper();
        printf("\n have finished starting the stepper");
        CloseSphere();
        j=0;
        outportb(0x108,0x8+6);
        delay(25);
        outportb(0x108,6);
        InitializeInterrupts();
        while(DarkDataFlag==0){

         if(ConvertTime){


             for(i=0;i<8;i++){

             outportb(0x108,0x8+6);
             k++;k--;  /* stall for a little time */
             outportb(0x108,6);    /* low gain conversion  */
             delay(1);

 ltemp[i]=(unsigned)(255-inportb(0x108))+(255-(inportb(0x109)<<8));
             outportb(0x108,0x8+5);
             k++;k--;   /* stall for a little time  */
             outportb(0x108,5);    /* high gain conversion  */
             delay(1);  /* */

htemp[i]=(unsigned)(255-inportb(0x108))+(255-(inportb(0x109)<<8));
             }

HighGainDarkData[j]=(unsigned)((htemp[0]+htemp[1]+htemp[2]+htemp[
3]+htemp[4]+htemp[5]+htemp[6]+htemp[7])>>3);

LowGainDarkData[j]=(unsigned)((ltemp[0]+ltemp[1]+ltemp[2]+ltemp[3
]+ltemp[4]+ltemp[5]+ltemp[6]+ltemp[7])>>3);
             j++;
             if(j==NumOfBins) {
             DarkDataFlag=1;
             }
             ConvertTime=0;

         }

        }
        AssembleDarkBlock();                          /*          */
        OpenSphere();

        CurrentCmdActive=1;
        LastGratingDiodeStatus=CurrentGratingDiodeStatus;
        CurrentGratingDiodeStatus=(inportb(0x10a)&1);

if((LastGratingDiodeStatus==0)&(CurrentGratingDiodeStatus==1)){
        BeginScanTime=1;
        }
        while(CurrentCmdActive){  /* loop until sequence is over*/
         LastGratingDiodeStatus=CurrentGratingDiodeStatus;
         CurrentGratingDiodeStatus=(inportb(0x10a)&1);

if((LastGratingDiodeStatus==0)&(CurrentGratingDiodeStatus==1)){
            InitializeInterrupts(); /*  */
            AssembleBlock();
            ScanCount++;
            if(ScanCount==CmdEntry[CmdEntryCount].NumOfScans){
              CurrentCmdActive=0;
              ScanCount=0;
            }
             if(ScanCount%2==0) {
/*           outportb(0x109,0); /* filter out */
             Filter=0;
             }
             if(ScanCount%2==1){
/*           outportb(0x109,0x80); /* filter in  */
             Filter=1;
             }

         }
      if(ConvertTime){
          if(CurrentGratingDiodeStatus==0){

             for(i=0;i<8;i++){
             outportb(0x108,0x18+6);
             k++;k--;             /* stall for a little time */
             outportb(0x108,0x10+6);    /* low gain conversion   */
             delay(1);

ltemp[i]=(255-inportb(0x108))+(255-(inportb(0x109)<<8));

             outportb(0x108,0x18+5);
             k++;k--;              /* stall for a little time  */
             outportb(0x108,0x10+5);    /* high gain conversion  */
             delay(1);

htemp[i]=(255-inportb(0x108))+(255-(inportb(0x109)<<8));
             }

                  if(!Filter){

                 L o w G a i n N o F i l t e r [ B i n C o u n t ] =
(unsigned)((ltemp[0]+ltemp[1]+ltemp[2]+ltemp[3]+ltemp[4]+ltemp[5]
+ltemp[6]+ltemp[7])>>3);

HighGainNoFilter[BinCount]=(unsigned)((htemp[0]+htemp[1]+htemp[2]
+htemp[3]+htemp[4]+htemp[5]+htemp[6]+htemp[7])>>3);

              } else {
                 L o w G a i n W i t h F i l t e r [ B i n C o u n t ] =
(unsigned)((ltemp[0]+ltemp[1]+ltemp[2]+ltemp[3]+ltemp[4]+ltemp[5]
+htemp[6]+htemp[7])>>3);

HighGainWithFilter[BinCount]=(unsigned)((htemp[0]+htemp[1]+htemp[
2]+htemp[3]+htemp[4]+htemp[5]+htemp[6]+htemp[7])>>3);
                }

             if( BinCount== 83) outportb(0x109,0x80);/* */
             BinCount++;
          }
          ConvertTime=0;
          /*  outportb(0x108,0); /* */
      }


        }

        printf("\n have finished assembling the data block");
        disable();
        StopStepper();
        WriteToFile();
        CmdEntryCount++;
        ScanCount=0;
        enable();
      }
    }
    fclose(fp2);
    fclose(fp3);
    CleanUp();
    printf("\n   two   numbers   of   interest   might   be   %u
  %u",temp[0],temp[1]);
    outportb(0x109,0);
    printf("\n That's All Folks!");

  }
```

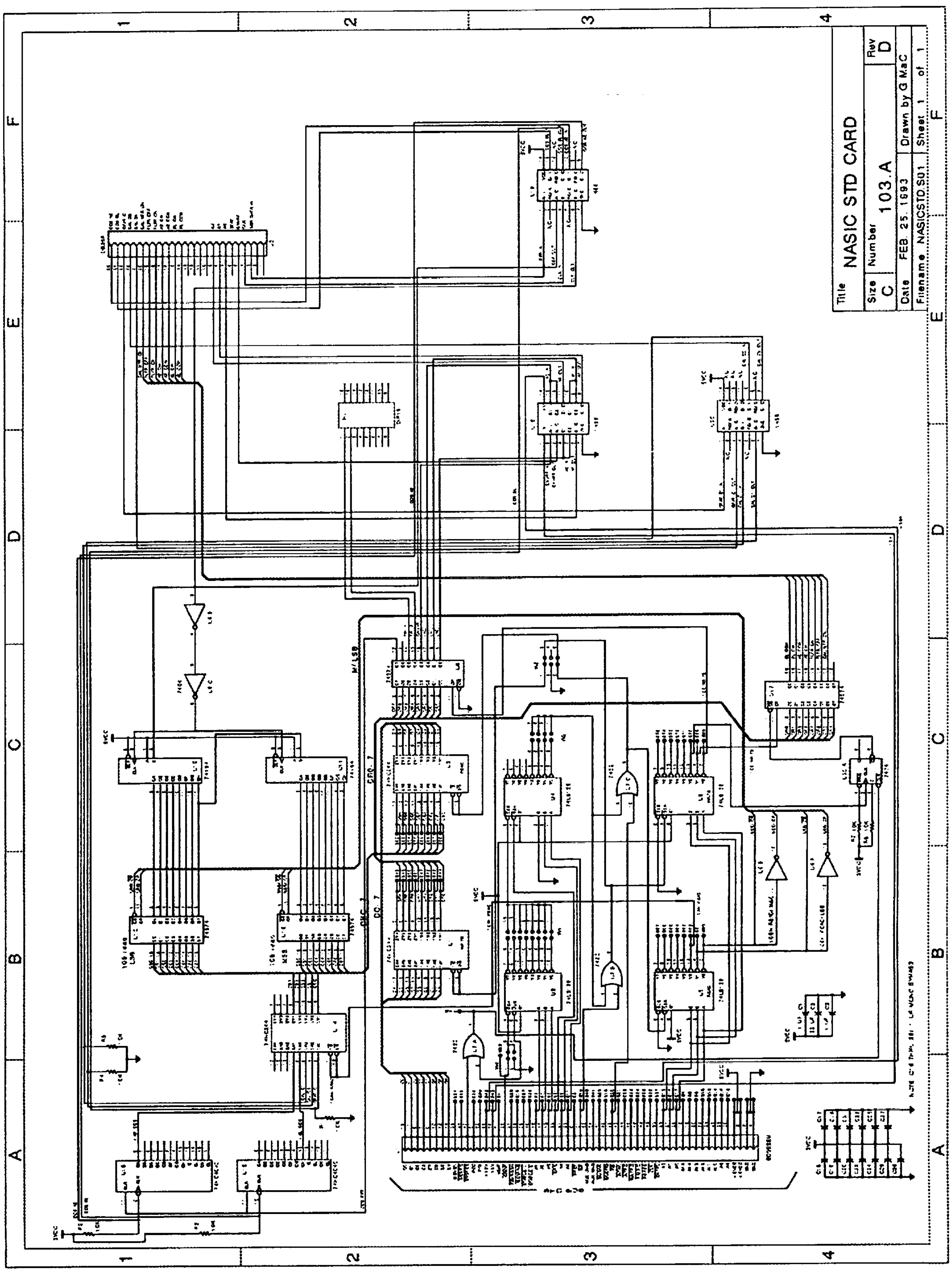
9. APPENDIX B - HARDWARE SCHEMATICS

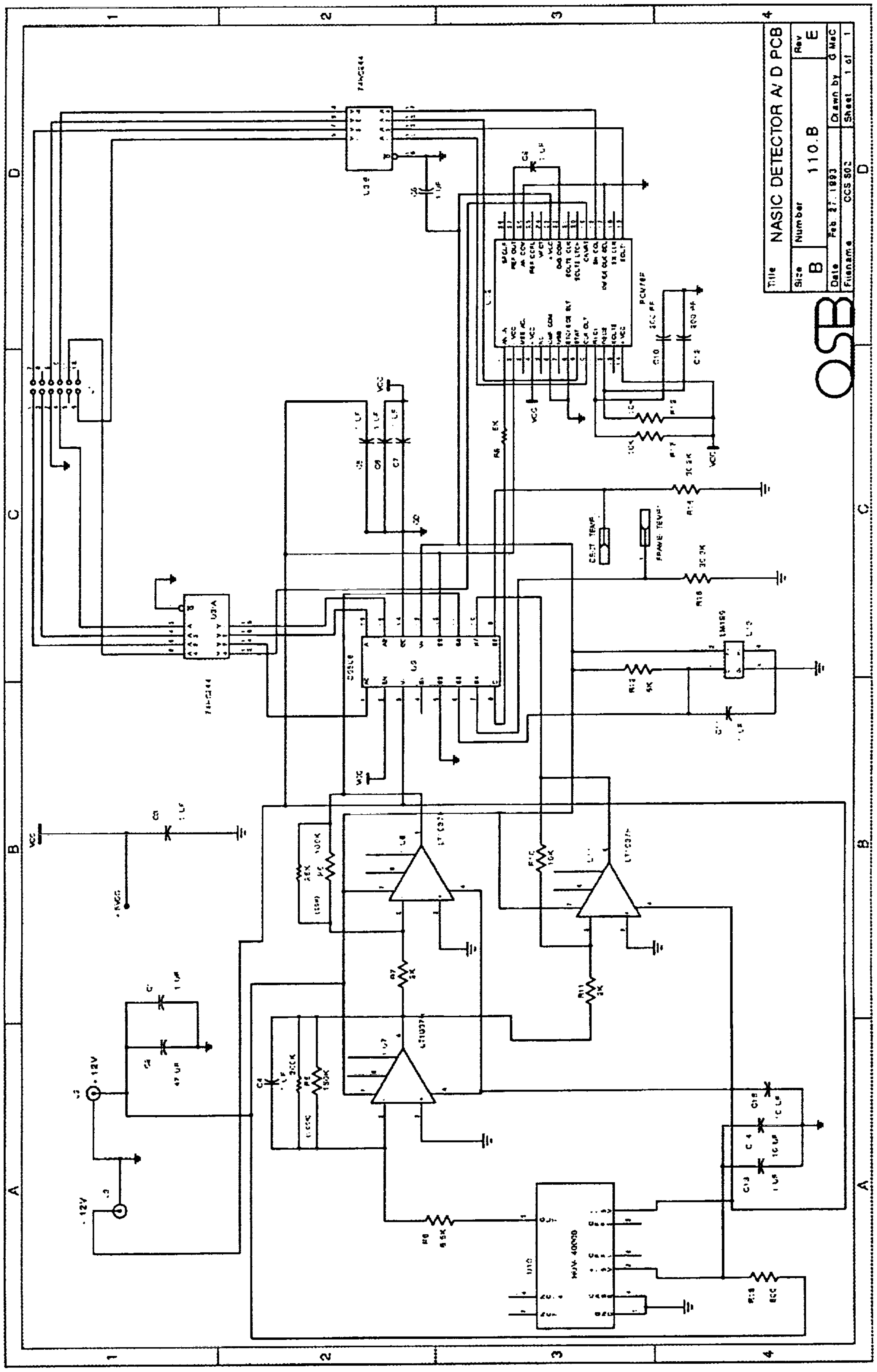

PDL- J1
EL CCW DR
EL CW DR
AZ CCW DR
AZ CW DR
FS ON
FS OFF
CS MTR ON
PDL GND
CAL D1
CAL D2
STPR A
GRAT D1
STPR B
CCS EL
CCS AZ
STPR C
STPR COM
DB25M
TLSCP- P4#5
TLSCP- P4#7
CS- P1#1
TLSCP- P2#4
TLSCP- P2#2
TLSCP- P2#3
TLSCP- P2#1
PDL- J2
28V
PDL- J3
SER OUT
CLK
CNVRT
STAT
A0
A1
A2
DB9M
+12VDC -12VDC
PDL- J4
TE- HI TE- LO
TLSCP- P4#1
TLSCP- P4#4
TLSCP- P4#2
TLSCP- P4#3
TM2 TM1
PDL- J5
TB1
28 V
TLSCP- P4#1
ELSW1
TB2
5VDC
AZSW1
AZSW2
AZENC
ELENC
COM/ BLK
+5V/ RED
COS/ GRN
BRP1
GND
A0 IN
A1 IN
EL CCS OUT
AZ CCS OUT
A2 IN
CNVRT IN
SER OUT
CLK OUT
STAT OUT
GRAT OUT
CAL D1 OUT
CAL D2 OUT
5VDC
A0 OUT> TLSCP- P3#5
A1 OUT> TLSCP- P3#6
EL CCS IN
AZ CCS IN
A2 OUT> TLSCP- P3#8
CNVRT OUT > TLSCP- P3#4
SER IN > TLSCP- P3#1
CLK IN > TLSCP- P3#10
STAT IN> TLSCP- P3#3
GRAT IN> TLSCP- P2#7
CAL D1 IN> CS- P1#3
-12VDC
CAL D2 IN> CS- P1#6
+12VDC
ELJ1
ELR1 40
ELR2 40
ELM1 EL MTR
ELSW2
AZJ1
AZR1 36
AZR2 36
AZM1 AZ MTR
OSB
OBSERVATIONAL
SCIENCE BRANCH
NASA/ WALLOPS FLIGHT FACILITY
Title NASIC TURRET CIRCUITS
Size A
Number 100.A
Rev D
Date FEB. 1, 1993
Drawn by GMaC
Filename NSCTUR.S01
Sheet 1 of 1

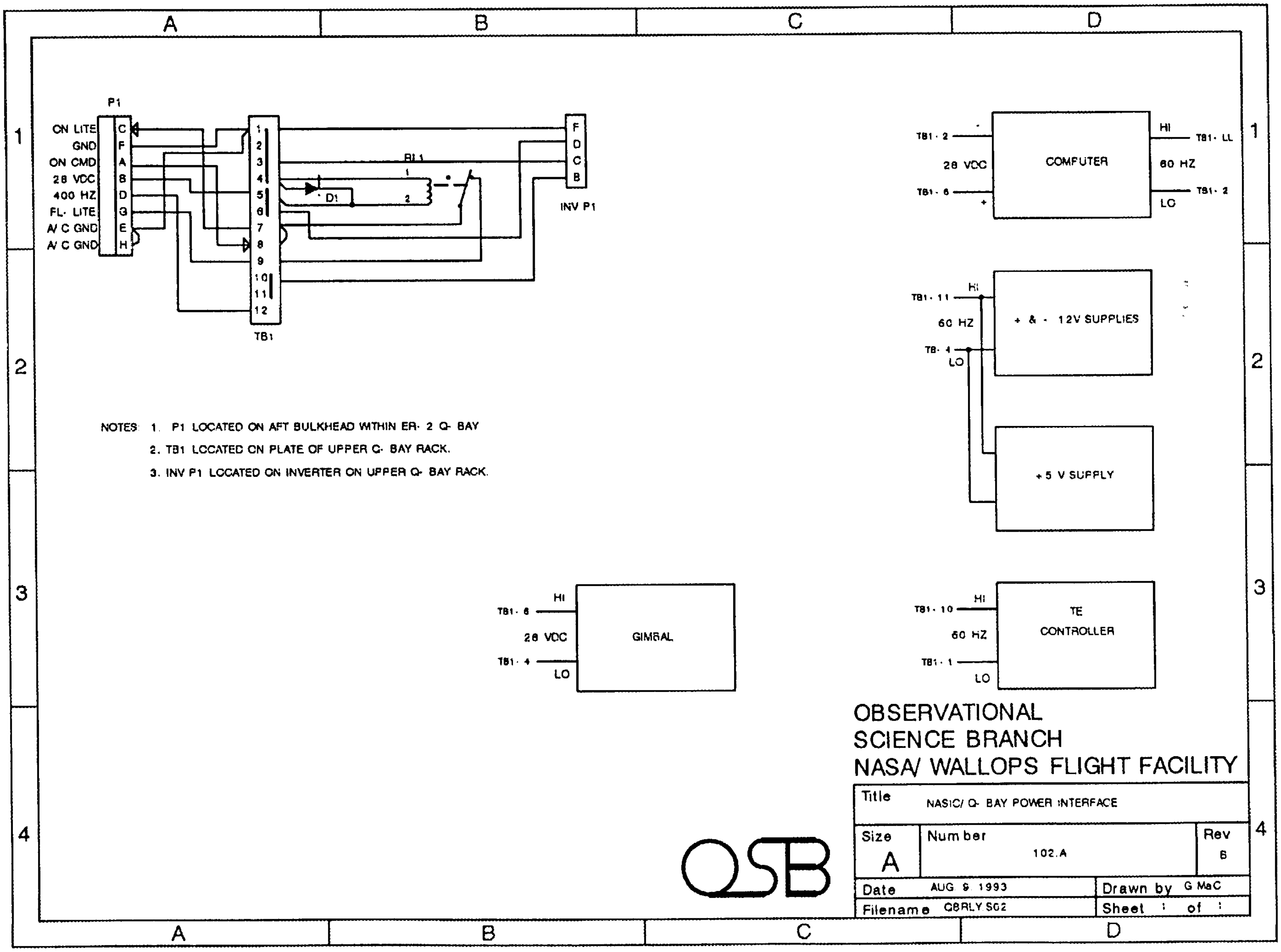

MOTOR DRIVER SWITCHING
Title
Size A
Number 104.A
Rev B
Date FEB 1, 1993
Drawn by G MaC
Filename NSCTRTRL.SC1
Sheet 1 of 1
OSB
EL CCW
EL CW
AZ CCW
AZ CW
FLTR SOL ON
FLTR SOL OFF
CAL SPHERE ON
RL1
RL2
RL3
RL4
RL5
RL6
RL7
+ OUT
INPUT
+5
5V- LO
RLYODV5M
+28V
ELR1 40
ELR2 40
AZR1 36
AZR2 36
ELMTR
AZMTR
FLTRSOLCCW
FLTRSOLCW
CALSPHRMTR
J1
DB9F
NOTE: THIS IS A SIMPLIFIED REPRESENTATION OF THE CIRCUIT.


TLSCP- P2
STPR COM
STPR B
STPR C
STPR A
GRAT +5V
GRAT LO
GRAT D1
DB15F
PDL/ TB2- 1
PDL GND
TLSCP- P3
SER OUT
GND
CNVRT
A0
A1
GND
A2
VCC 5V
CLK
DB15M
PDL GND
PDL GND
PDL/ TB2- 1
TLSCP- P4
28V
GND
FS ON
FILTER LO
FS OFF
DB9M
PDL/ TB1- 2
PDL GND
PDL GND
CS- P1
CS MTR ON
+5V
CS D1 OUT
DB9F
PDL/ TB2- 1
BRP1
GND
A0 IN
A1 IN
EL COS OUT
AZ COS OUT
A2 IN
CONV IN
SER OUT
CLK OUT
STAT OUT
GRAT OUT
CAL D1 OUT
VCC 5V
CAL D2 OUT
GND
A0 OUT
A1 OUT
EL COS IN
AZ COS IN
A2 OUT
CONV OUT
SER IN
CLK OUT
GRAT IN
CAL D1 IN
VCC 5V
GND
VCC
DB25M
STPR COM
STPR C
COS AZ
COS EL
STPR B
GRAT D1
STPR A
CS D2 OUT
CS D1 OUT
PDL GND
CS MTR ON
FS OFF
FS ON
AZ CW DR
AZ CCW DR
EL CW DR
EL CCW DR
PDL GND
TB2- 5
TB2- 6
ELSW2- 1
TB1- 1
PDL- J1
DB9M
SER OUT
CLK
CNVRT
STAT
A0
A1
A2
PDL- J3
OBSERVATIONAL
SCIENCE BRANCH
NASA/ WALLOPS FLIGHT FACILITY
Title
NASIC TURRET/ TELESCOPE INTERFACE CABLE CABLE
Size A
Number 106.A
Rev D
Date FEB. 1, 1993
Drawn by GMaC
Filename NSCCBL1.S01
Sheet 1 of 1


OBSERVATIONAL
SCIENCE BRANCH
NASA/ WALLOPS FLIGHT FACILITY
Title TURRET BUFFER BOARD
Size A
Number 105.A
Rev C
Date FEB. 1, 1993
Drawn by GMaC
Filename NSC232.S01
Sheet 1 of 1
J1
U1 1489
U2 1488
U3 1488
U4:A 7432
U4:B 7432
5VDC
+12VDC
-12VDC
C1 .1 UF
C2 47 UF
C3 .1 UF
C4 47 UF
C5 .1 UF
C6 47 UF
R1 47
R2 47
R3 47
R4 47
R5 1K
R6 1K
R7 10K
R8 10K
R9 10K
GD


Title
NASIC STD CARD
Size
C
Number
103.A
Rev
D
Date
FEB. 25. 1993
Drawn by G MaC
Filename NASICSTD.S01
Sheet 1 of 1
A
B
C
D
E
F
1
2
3
4


Title NASIC DETECTOR A/ D PCB
Size B
Number 110.B
Rev E
Date Feb. 27, 1993
Drawn by G MaC
Filename CCS 502
Sheet 1 of 1
OSB
74HC244
U3A
U9
PCM78P
LM199
LT1037
NJV 4000
CELL TEMP
FRAME TEMP
+12V
-12V
VCC
+5VCC

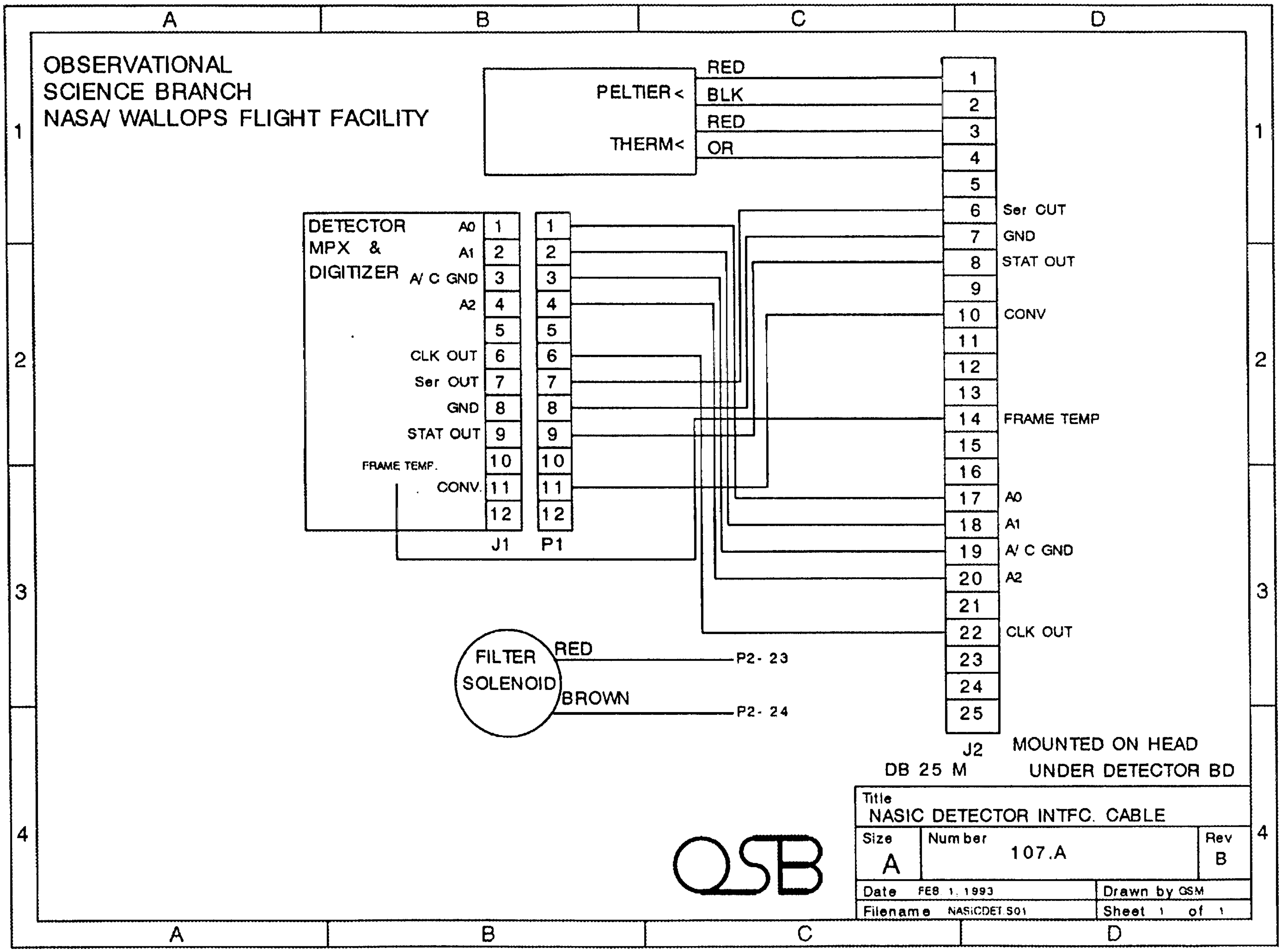


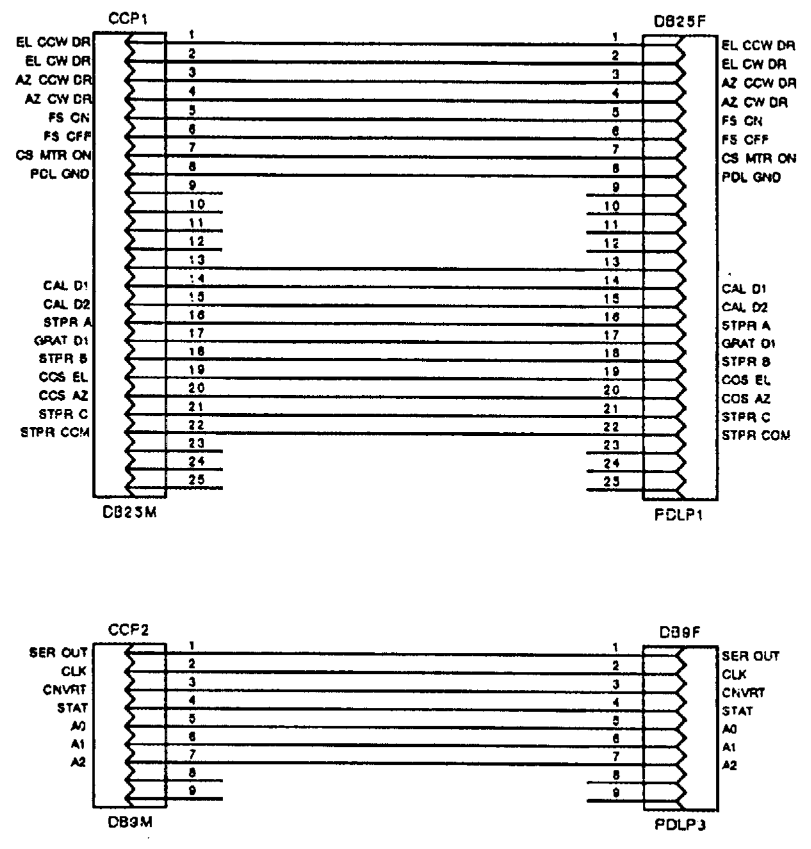
NOTES: 1. P1 LOCATED ON AFT BULKHEAD WITHIN ER- 2 Q- BAY
2. TB1 LOCATED ON PLATE OF UPPER Q- BAY RACK.
3. INV P1 LOCATED ON INVERTER ON UPPER Q- BAY RACK.

OBSERVATIONAL
SCIENCE BRANCH
NASA/ WALLOPS FLIGHT FACILITY

OSB

Title	NASIC/ Q- BAY POWER INTERFACE	
Size A	Number 102.A	Rev B
Date AUG 9 1993		Drawn by G MaC
Filename Q8RLY.S02		Sheet 1 of 1

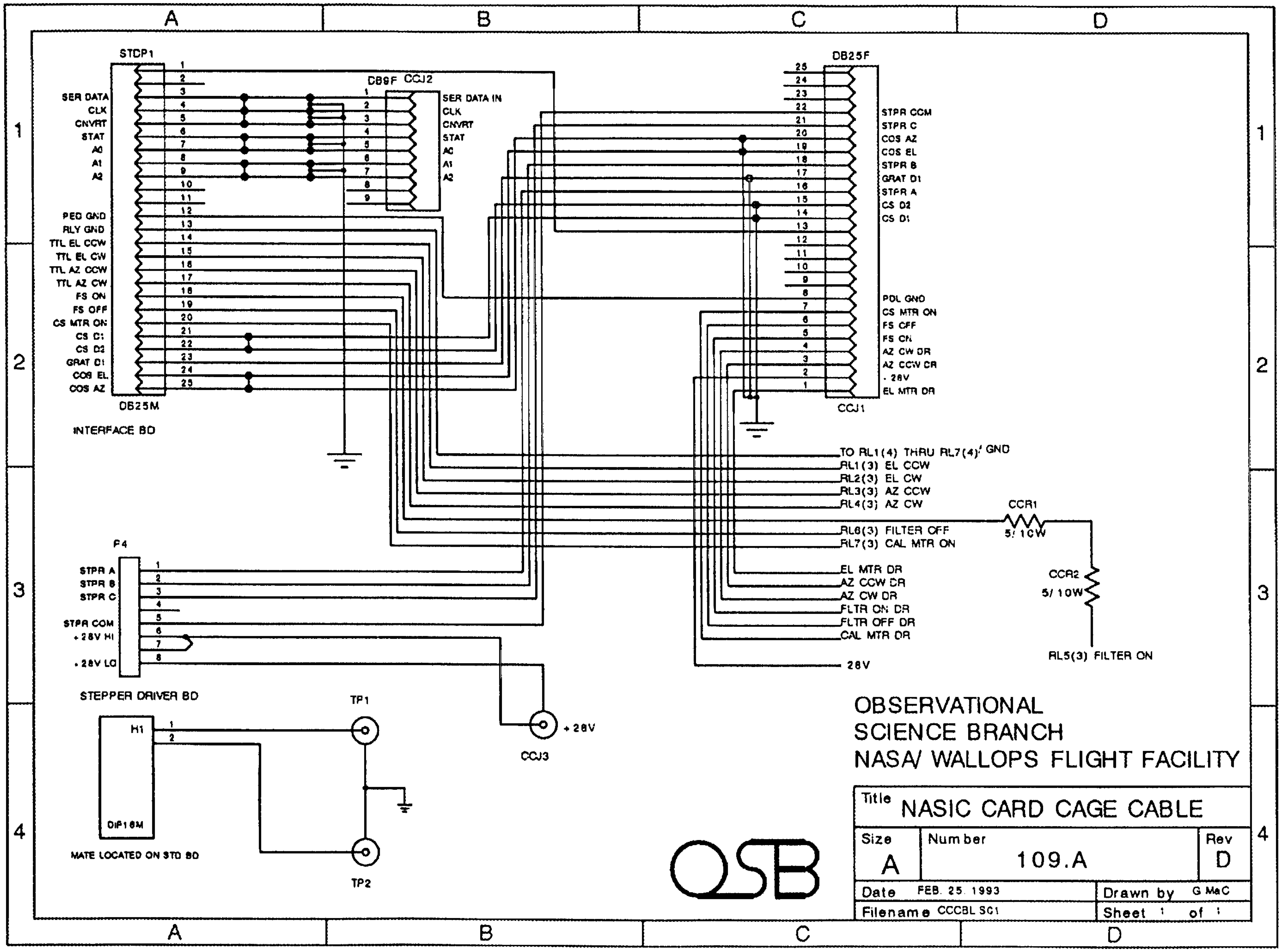

OBSERVATIONAL
SCIENCE BRANCH
NASA/ WALLOPS FLIGHT FACILITY
PELTIER<
THERM<
RED
BLK
RED
OR
DETECTOR
MPX &
DIGITIZER
A0
A1
A/ C GND
A2
CLK OUT
Ser OUT
GND
STAT OUT
FRAME TEMP.
CONV.
J1
P1
Ser OUT
GND
STAT OUT
CONV
FRAME TEMP
A0
A1
A/ C GND
A2
CLK OUT
J2
DB 25 M
MOUNTED ON HEAD
UNDER DETECTOR BD
FILTER
SOLENOID
RED
BROWN
P2- 23
P2- 24
OSB
Title
NASIC DETECTOR INTFC. CABLE
Size
A
Number
107.A
Rev
B
Date FEB 1, 1993
Drawn by GSM
Filename NASICDET.S01
Sheet 1 of 1



OBSERVATIONAL
SCIENCE BRANCH
NASA/ WALLOPS FLIGHT FACILITY

OSB

Title: NASIC CARD CAGE / TURRET INTERFACE CABLE		
Size: A	Number: 108.A	Rev: D
Date: FEB.1, 1993	Drawn by GMaC	
Filename: CCTOTUR.S01	Sheet 1 of 1	


STDP1
SER DATA
CLK
CNVRT
STAT
A0
A1
A2
PED GND
RLY GND
TTL EL CCW
TTL EL CW
TTL AZ CCW
TTL AZ CW
FS ON
FS OFF
CS MTR ON
CS D1
CS D2
GRAT D1
COS EL
COS AZ
DB25M
INTERFACE BD
DB9F CCJ2
SER DATA IN
CLK
CNVRT
STAT
A0
A1
A2
DB25F
STPR COM
STPR C
COS AZ
COS EL
STPR B
GRAT D1
STPR A
CS D2
CS D1
PDL GND
CS MTR ON
FS OFF
FS ON
AZ CW DR
AZ CCW DR
- 28V
EL MTR DR
CCJ1
TO RL1(4) THRU RL7(4)/ GND
RL1(3) EL CCW
RL2(3) EL CW
RL3(3) AZ CCW
RL4(3) AZ CW
RL6(3) FILTER OFF
RL7(3) CAL MTR ON
EL MTR DR
AZ CCW DR
AZ CW DR
FLTR ON DR
FLTR OFF DR
CAL MTR DR
28V
CCR1
5/ 10W
CCR2
5/ 10W
RL5(3) FILTER ON
P4
STPR A
STPR B
STPR C
STPR COM
+ 28V HI
+ 28V LO
STEPPER DRIVER BD
H1
DIP16M
MATE LOCATED ON STD BD
TP1
TP2
CCJ3
+ 28V
OBSERVATIONAL
SCIENCE BRANCH
NASA/ WALLOPS FLIGHT FACILITY
OSB
Title NASIC CARD CAGE CABLE
Size A
Number 109.A
Rev D
Date FEB. 25. 1993
Drawn by G MaC
Filename CCCBL SC1
Sheet 1 of 1

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE July 1993	3. REPORT TYPE AND DATES COVERED Reference Publication

4. TITLE AND SUBTITLE NASA Airborne Satellite Instrumentation Calibrator (NASIC) Technical Reference	5. FUNDING NUMBERS Code 972
6. AUTHOR(S) John L. Ward and Gerry McIntire	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Wallops Flight Facility Wallops Island, Virginia 23337	8. PERFORMING ORGANIZATION REPORT NUMBER 93B00107
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, D.C. 20546–0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA RP–1315

11. SUPPLEMENTARY NOTES

Gerry McIntire: Computer Sciences Corporation, Wallops Island, Virginia, 23337.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified–Unlimited Subject Category 19 Report available from the NASA Center for AeroSpace Information, 800 Elkridge Landing Road, Linthicum Heights, MD 21090; (301) 621–0390.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The NASA Satellite Instrumentation Calibrator (NASIC) is a visible and near–infrared spectrometer used to calibrate various satellite instruments by underflying those instruments in a NASA ER–2 aircraft. This report documents the calibration instrument's hardware and software.

14. SUBJECT TERMS Satellite Instrumentation, NASIC, Instrument Calibration, ER–2	15. NUMBER OF PAGES 47
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited